Club des Utilisateurs de Micro-ordinateurs dans l'Education

# Stage Virtualisation Serveurs


Juin 2008

Xavier Montagutelli
Université de Limoges
Service Commun Informatique
xavier.montagutelli@unilim.fr

Hubert Chomette
Université de Limoges
École Nationale Supérieure d'Ingénieurs de Limoges
hubert.chomette@unilim.fr

# Licence

# Plan

- Pourquoi virtualiser ?
- Techniques de virtualisation
- Linux VServer
  - Architecture
  - Mise en œuvre
  - Retour d’expérience
- Conclusions & perspectives
- TP

# Plan

- Pourquoi virtualiser ?
- Techniques de virtualisation
- Linux VServer
  - Architecture
  - Mise en œuvre
  - Retour d’expérience
- Conclusions & perspectives
- TP

# Pourquoi virtualiser ? Etat des lieux

- Prolifération des serveurs
  - ↗ Quantité et traitements de données numériques
  - 1 application = 1 serveur : incompatibilités, facilité, sécurité, simplification de l'administration, contrainte des éditeurs
  - Tests et développements
  - Redondance
- ⇨ Salles techniques saturées, sous-dimensionnées (climatisation, protection électrique)
- ⇨ Matériel actif (réseau, SAN) ↗
- ⇨ Temps perdu en « logistique » : commande des machines, réception, installation, déploiement OS
- ⇨ Matériel pas homogène

# Pourquoi virtualiser ? Etat des lieux (2)

- Machines de plus en plus puissante (en CPU, en mémoire)
- ⇨ Sous-utilisation

- Bilan : coûts financiers et humains élevés, et une perte de ressources
- La virtualisation : un outil de **consolidation** (regroupement des ressources pour optimiser leur administration / utilisation)
  Exemple : stockage, serveurs

# Pourquoi virtualiser ? Objectifs

- Réduction des coûts (matériels, maintenance, …)
- Amélioration du niveau de service et flexibilité
- Renforcement de la sécurité
- Simplification de l’administration

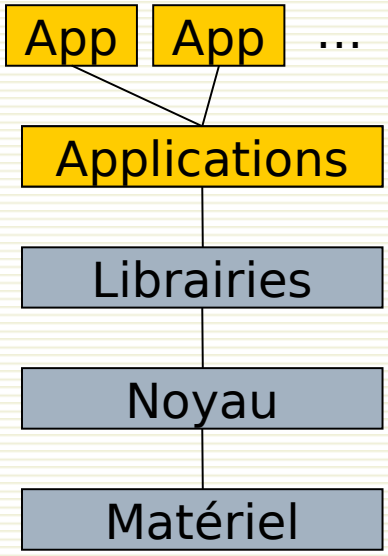
# Pourquoi virtualiser ? Une évolution forte

- Beaucoup d'articles
  - January 9, 2006 - Network World
    Virtualization a Hot Technology for 2006
- Banalisation / démocratisation des solutions
  - Intégré dans les distributions Linux (Fedora, Redhat, Debian, Mandriva)
  - Intégré dans Windows, gratuit (virtual server ou Hyper-V)
    http://www.microsoft.com/windowsserver2008/en/us/virtualization-consolidation.aspx
  - February 6, 2006 : VMware Introduces Free VMware Server

# Plan

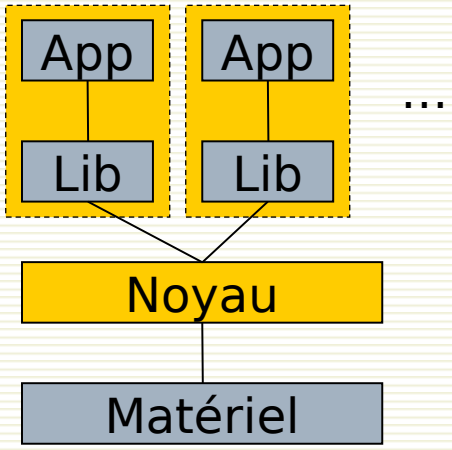
- Pourquoi virtualiser ?
- Techniques de virtualisation
- Linux VServer
  - Architecture
  - Mise en œuvre
  - Retour d’expérience
- Conclusions & perspectives
- TP

# Techniques de virtualisation – Niveau applicatif




- La virtualisation peut intervenir à différents niveaux
- Au niveau applicatif : l'application fait croire qu'il y a plusieurs services
  - Hôtes virtuels Apache, domaines virtuels Postfix, ...
  - Performances optimales

# Techniques de virtualisation – Conteneur

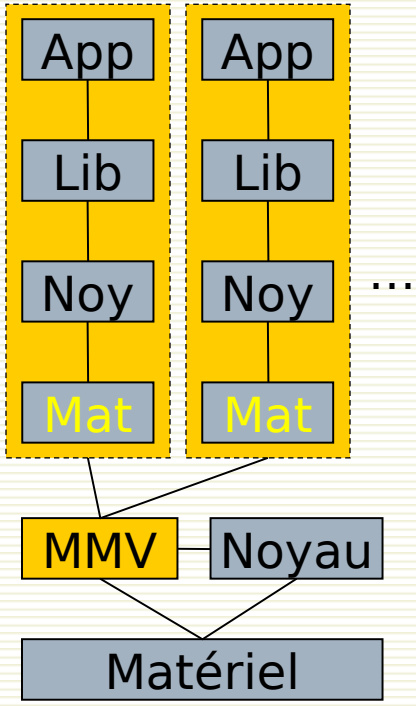



- Au niveau du noyau : **séparation des applications**, regroupées dans des « cages » étanches
  - Un seul noyau, qui fait croire à plusieurs machines
  - Il répartit les ressources
  - BSD Jails, Solaris Zones, **Linux VServer**, OpenVZ
  - Performances excellentes

# Techniques de virtualisation – Conteneur Linux

- Pas de solution « native » et complète sous Linux
- Des projets anciens : Linux-VServer, OpenVZ
- Des composants déjà existant dans le noyau (chroot, capacités, VFS namespace)
- Un intérêt croissant et une volonté d'intégrer des composants manquants dans le noyau
  - 2.6.18 : *UTS namespace*
  - 2.6.24 : *PID namespace*
  - 2.6.24 : *control groups* (cgroups), nommé dans un premier temps *process container*
  - 2.6.24 : *network namespace*
- Des éléments qui pourraient servir pour du checkpoint/restart, de la migration de processus, des politiques d'ordonnancement par usager, …

# Techniques de virtualisation – Hyperviseur




- Un *hyperviseur* (de type 1), ou *moniteur de machine virtuelle*, fonctionne directement au-dessus du matériel
  - VMware ESX, Microsoft Hyper-V, **Xen**
  - Les instructions machines s'exécutent en grande partie nativement sur le processeur
  - Performances bonnes à très bonnes
  - NB : un hypercall ou appel hyperviseur signifie que le système invité fait directement un appel à l'hyperviseur (par référence aux appels systèmes, syscall) pour exécuter des instructions *sensibles*

# Techniques de virtualisation – Hyperviseur x86

- Le processeur x86 « classique » se prête mal à la virtualisation, malgré son mode dit *protégé* qui offre 4 niveaux de privilèges différents (ring 0 réservé au noyau, ring 3 pour les applications)
- L'hyperviseur et la technique des hypercall sont là pour pallier ses carences : allocation mémoire, interception (ou détournement par hypercall) des instructions processeurs sensibles
- Les technologies Intel-VT / AMD SVM (ou AMD-V) introduites en 2006 ont rendu le processeur « virtualisable », ce qui allège le travail de l'hyperviseur
- Très bientôt, le matériel apportera aussi la virtualisation pour la mémoire

# Plan

- Pourquoi virtualiser ?
- Techniques de virtualisation
- Linux VServer
  - Architecture
  - Mise en œuvre
  - Retour d’expérience
- Conclusions & perspectives
- TP

# Linux VServer – Introduction (1)

- Idée : séparer l'espace utilisateur d'un système GNU/Linux (« hôte ») en unités distinctes (« serveurs privés virtuels » ou « vservers »)
- Patch sur le noyau Linux (`patch-<version_linux>-vs<version_vserver>`)
- Commandes utilisateurs (`util-vserver`)
- http://linux-vserver.org/
- Liste de diffusion http://list.linux-vserver.org
- **#vserver** sur **irc.oftc.net**
- Début du projet en 2001, utilisable depuis 2003

# Linux VServer – Introduction (2)

- Historique
  - Liste de diffusion : 2001
  - Version 1.0, novembre 2003 (Linux 2.4.20)
    patch 1146 lignes ajoutées ou modifiées
  - **Version 1.2, décembre 2003 → janvier 2005**
    patch > 2000 lignes
  - **Version 2.0, août 2005 (Linux 2.6.12)**
    patch ≈ 10000 lignes
  - Version 2.2.0, nov. 2006 (Linux 2.6.20)
  - **Version 2.2.0.7, mars 2008 (Linux 2.6.22.19)**
    patch ≈ 17000 lignes, 458 fichiers
  - Adaptation en cours aux 2.6.24+ avec patchs
    « containers »

# VServer – Isolation de processus

- **Contexte** : nouvelle structure du noyau, identifié par un entier
- Chaque processus fait partie d'un contexte
- Interactions entre processus (signaux, IPC…) limitées à un contexte ($\Rightarrow$ *isolation* plutôt que *virtualisation*)
- Contexte de l'hôte : 0
  - Peut créer de nouveaux contextes
  - Peut changer de contexte
- Contexte « spectateur » : 1
  - Peut voir les processus de tous les contextes
- Un contexte $\approx$ un vserver

# VServer – Isolation réseau

- L'hôte dispose de plusieurs adresses réseaux (éventuellement des alias)
- Les processus d'un vserver sont limités à une (ou plusieurs) adresse(s). Plus précisément, les processus vont être rattachés à un « *network context* » (introduits dans VServer 2.2)
- Attention, les applications de l'hôte doivent être « bindées » sur l'adresse IP qui lui est dédiée

# VServer – Isolation système de fichiers

- Chroot : la racine apparente du FS (/) est en réalité un répertoire de l'hôte
- Nouvel attribut du système de fichiers pour se prémunir de l'évasion (barrier)
- Utilisation des espaces de noms (namespaces) de la couche VFS : chaque VServer a son namespace et une vue différente du FS
- Possibilité d'associer un fichier a un contexte
  - Clé d'accès
  - Nécessaire pour avoir une limite disque par VServer et des quotas par VServer dans le cas d'une partition partagée

# VServer – Limitation du super-utilisateur

- Capacités
  - Brouillon de norme POSIX, partiellement supportée depuis Linux 2.2
  - Jeton présenté par un processus pour prouver qu'il est autorisé à faire une action
  - Exemple : créer un fichier périphérique (MKNOD)
  - On peut fixer une limite aux capacités d'un contexte ⇒ *root* ne pourra pas tout faire (*bounding capabilities*, bcaps)
- Nouvelles capacités (*context capabilities*, *ccaps*). Exemple :
  - CAP_NET_RAW trop fort. Mais sans lui, pas de ping…
  - Solution : VXC_RAW_ICMP

# VServer – Isolation et extension de /proc

- /proc
  - Système de fichiers virtuel
  - Accès (lecture ou lecture/écriture) aux informations du noyau
- Nécessaire dans un vserver : uptime, liste des processus, type de cpu, mémoire utilisée, points de montage, ...
- Mais pas tout
  - Les processus des autres contextes n'apparaissent pas
  - Certaines entrées sont « cachées » à l'aide d'attributs supplémentaires
- Extensions sous /proc/virtual/ et /proc/virtnet/

# VServer – Limiter les ressources

- Coopération des processus et allocation des ressources : par le noyau (classiquement)
- ulimit par vserver : limitation de la mémoire, du nombre de processus, ...
- Consommation de CPU limitable par un algorithme « seau de jeton »
- Disque
  - Une partition par vserver...
  - ... ou utiliser le marquage des fichiers par contexte

# VServer – Autres éléments

- Virtualisation d'informations systèmes
  - Nom d'hôte, version et release d'OS, type de machine, processeur (*utsname*, uname –a)
  - Uptime
  - Quantité de mémoire disponible (en fonction des limites fixées)
- Unification
  - Partager des fichiers entre VServer à travers des liens en dur, idéalement tout / sauf …
  - Gain de disque
  - Mise à jour

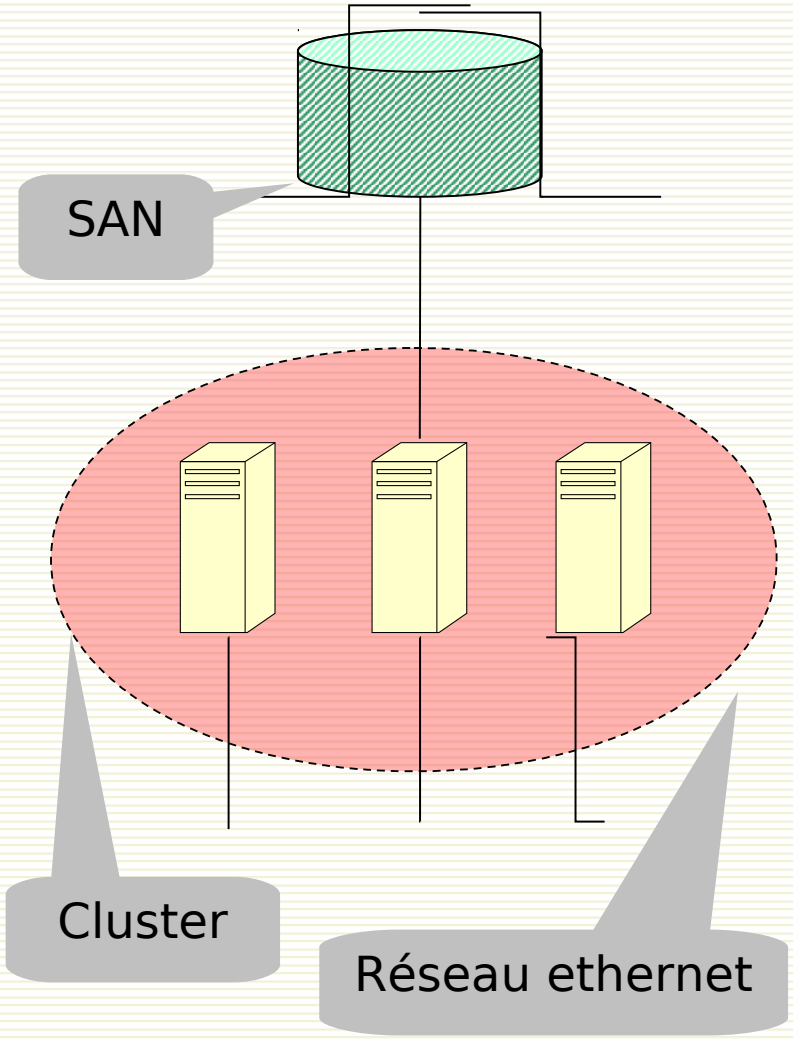
# VServer – Limites

- Interface de boucle locale : pseudo-loopback dans la branche de développement 2.3
- Support IPv6 : dans 2.3
- NFS en mode noyau : non
- Migration à chaud de VServer : non
- Des vservers sur plusieurs VLAN : possible
- Netfilter par VServer : non, nécessite une virtualisation complète de la couche réseau
- Manque une interface qui simplifierait la gestion
- Nécessité de bien connaître GNU/Linux et compréhension des mécanismes ci-dessus…

# VServer – Retour d’expérience

- INSA de Toulouse : version 1.2 depuis 2004
  - P4, RAM 1Go, LVM sur disques locaux en miroirs
  - Serveur www institutionnel
  - 8 autres serveurs (applications web), pour répondre à des demandes rapidement
- Université de Limoges : version 2.x depuis septembre 2005
  - 3 serveurs bi-Xeon, RAM 8 Go
  - Serveurs de courrier des étudiants (14000), serveur web institutionnel, webmail, ENT, DNS secondaire, moodle, FTP, …

# VServer – Retour d'expérience (2)




- Architecture en production à l'université de Limoges (janvier 2008)
- 3 serveurs (bi-Xeon EM64T, RAM 8 Go) formant un cluster Red Hat
- Un disque (du SAN, 700 Go) accessible à tous les membres
- Disque intégré dans un *Volume Group* Cluster LVM
- Pour migrer (à froid) les VServers d'un hôte vers un autre
- Avec surveillance automatique des hôtes et des VServers (haute-dispo)

# Plan

- Pourquoi virtualiser ?
- Techniques de virtualisation
- Linux VServer
  - Architecture
  - Mise en œuvre
  - Retour d’expérience
- Conclusions & perspectives
- TP

# Conclusions & perspectives

- ☺ Une solution qui tient ses promesses : efficace, léger, robuste
- ☹ Après son apprentissage ...
- ☹ Manque d'intégration dans les distributions
- ☹ Des points à améliorer dans ou autour de VServer
  - Outils de supervision
  - Outils d'administration
- ➲ Approche de consolidation à intégrer dans une démarche globale ?

# Pages importantes

- http://2005.jres.org/paper/109.pdf
- http://www.renater.fr/Video/JRES/TutoJRESMars2008/P/Perrot/techvirtualisation-tutojre

- http://linux-vserver.org/Paper
- http://linux-vserver.org/Feature_Matrix
- http://linux-vserver.org/Frequently_Asked_Questions
- http://linux-vserver.org/Capabilities_and_Flags
- http://linux-vserver.org/CPU_Scheduler
- http://linux-vserver.org/Resource_Limits
- http://linux-vserver.org/Memory_Limits
- http://www.nongnu.org/util-vserver/doc/conf/configuration.html

# Vserver : mise en oeuvre sous Linux Lenny



*D'après un document de :*

Xavier Montagutelli
Université de Limoges
Service Commun Informatique
xavier.montagutelli@unilim.fr

Hubert Chomette
Université de Limoges
École Nationale Supérieure d'Ingénieurs de Limoges
hubert.chomette@unilim.fr

Modification et mise à jour pour Debian squeeze:
P. Petit (petit@info.univ-evry.fr), 12/2011

# TP – Plan

- Prise en main de l’hôte Debian
- Installation des composants
- Installer son premier VServer
- Dupliquer un VServer
- Pour aller plus loin …

# Prise en main de l’hôte Debian

- Objectif : prendre en main la machine de TP, connaître les principales commandes propres à la distribution GNU/Linux Debian

# Prise en main

- Login debian, mot de passe debian
- ou login root, mot de passe passdebian
- Passer root dans un terminal :
  « su – root », mot de passe passdebian
- Distribution GNU/Linux Debian stable (*squeeze*)
- Chercher un paquet à l'aide d'un mot clé
  # **apt-cache search *mot_clé***
- Exemple
  # **apt-cache search vserver**
- Installer un paquet
  # **apt-get install *nom_paquet***

# TP – Plan

- Prise en main de l’hôte Debian
- Installation des composants
- Installer son premier VServer
- Dupliquer un VServer
- Pour aller plus loin …

# Installation des composants

- Pré-requis : avoir installé la distribution GNU/Linux Debian version «squeeze» sur un serveur. Cette installation devrait être « minimaliste » (seulement  des services réseaux strictement nécessaires afin de sécuriser au mieux le serveur)
- Objectif : installer les composants de Linux VServer pour transformer le serveur en hôte de VServers

# Composants de base - Noyau

- **Etape 1** : le noyau modifié VServer. Deux possibilités :
  - Soit avec les sources du noyau Linux : téléchargement des sources du noyau, du patch Linux-VServer, application du patch, configuration, compilation
  - Soit en utilisant un noyau « pré-packagé » : Debian en fournit un

→ Installation du paquet Debian :

```
# apt-get install linux-image-vserver-amd64
```

→ Notez les paquets recommandés (mais dont l'installation n'est pas rendue obligatoire)

→ Rebooter

```
# reboot
```

! Choisir le noyau vserver au moment du boot

# Composants de base – Noyau (2)

- Quelle version de VServer est installée ? Le paquet Debian ne dit pas grand-chose …

→ Recherche dans la description du paquet :

```
# dpkg -s linux-image-vserver-amd64
```

→ Recherche dans le README Debian :

```
# zless /usr/share/doc/linux-image-2.6.18-6-vserver-amd64/debian.README.gz
```

→ Recherche dans le « changelog » Debian :

```
# zgrep -i vserver /usr/share/doc/linux-image-2.6.18-6-vserver-
    amd64/changelog.Debian.gz
```

- Ce n’est pas la dernière version de VServer !

# Composants de base – Commandes utilisateurs

- **Etape 2** : les commandes « userspace » officielles s'appellent util-vserver, empaquetées par Debian

→ vérifier que le paquet est installé

```
# dpkg –list |grep util-vserver
```

! Ce paquet est modifié par Debian par rapport à l'original

→ Vérifier le contenu (liste des fichiers) du paquet

```
# dpkg –L util-vserver
```

→ Un script d'amorçage `/etc/init.d/util-vserver`
→ Un répertoire `/etc/vservers`
→ Des commandes pour root (sous `/usr/sbin`) : `vserver`, `vps`, `vtop` …
→ Un répertoire `/usr/lib/util-vserver/`
→ Des pages de manuel
→ Un répertoire de doc /usr/share/doc/util-vserver

# TP – Plan

- Prise en main de l’hôte Debian
- Installation des composants
- Installer son premier VServer
- Dupliquer un VServer
- Pour aller plus loin …

# Installer son premier VServer

- Pré-requis :
  - Avoir installé la distribution GNU/Linux Debian version «squeeze» sur un serveur
  - Avoir installé les composants Linux VServer
- Objectif :
  - Installer un VServer basé sur Debian
  - Démarrer le VServer
  - Paramétrer le VServer
  - Voir l’état du VServer
  - Placer le VServer dans une partition dédiée
  - Rendre son démarrage automatique

# Construire un VServer

→ Installer le paquet *debootstrap*. debootstrap est une commande Debian qui permet d'installer les paquets minimums formant une machine Debian. Elle sera utilisée pour créer le répertoire du chroot du VServer. Vérifier que le paquet est installé :

```
# dpkg –list |gre  debootstrap
```

→ Construire le vserver, avec la commande **vserver**, qui est la commande de base pour interagir avec les VServers

- Argument 1 : nom du VServer
- Argument 2 : action, ici `build` pour construire un nouveau VServer
- -m <méthode> : méthode à utiliser pour construire le VServer
- --context <num> : assigne un numéro de contexte (de 2 à 32767), statique (assigné par l'administrateur), au VServer (les numéros supérieur et jusqu'à 65535 sont utilisés pour les contextes dynamiques)
- --hostname <nom_hote>
- --interface ... : indique l'interface réseau utilisée par le VServer, son adresse IP et son masque réseau
- -- : indique le début des options spécifiques à la méthode de build choisie
- -d <distrib> : nom de la distribution Debian que debootstrap doit installer
- Pour plus de détails sur la commande vserver, utilisez l'option –help (« `vserver –help` » ou « `vserver bidon <action> --help` » pour l'aide sur <*action*>)
- dans l'exemple ci-dessous, remplacer l'ip par une ip compatible avec votre réseau NAT vmware de façon à pouvoir accèder à internet :

```
# vserver monvs build \
    -m debootstrap --context 100 \
    --hostname monvs.cume.fr \
    --interface eth0:192.168.44.150/24 \
    -- -d squeeze

vserver monvs build --help
```

# Démarrer le VServer

- Le démarrage d'un VServer consiste (entre autres) à créer un alias sur l'interface réseau, à faire un chroot dans son arborescence, puis à lancer les services du VServer dans un contexte qui va les isoler des autres processus
- Plus de détails plus loin

→ « Démarrer » le VServer :

```
# vserver monvs start
```

# « Voir » le VServer

→ Lister les VServers actifs (= contextes existant)

```
# vserver-stat
CTX   PROC    VSZ    RSS  userTIME   sysTIME    UPTIME NAME
0       32  36.8M  11.5M   0m01s59   0m18s89  14m04s90 root server
100      2   3.7M   1.2M   0m00s00   0m00s10   0m02s20 monvs
```

- CTX : numéro de contexte
- PROC : nombre de processus
- VSZ, RSS : mémoire virtuelle et résidente (somme des colonnes VSZ et RSS d'une commande ps pour tous les processus des VServers)
  - ⓘ VSZ est surestimé à cause des pages partagées entre processus
- userTIME, sysTIME : temps user et système cumulé de tous les processus des VServers

→ Notez la consommation très faible de ressources

# « Voir » le VServer (2)

→ Lister les processus de tous les VServers

```
# vps auxw | more
```

→ Noter la colonne `CONTEXT`

→ Faire un top avec les processus des VServers

```
# vtop
```

→ Noter l'absence de colonne `CONTEXT` ...

→ Voir toutes les interfaces réseaux

```
# ip addr ls
# ifconfig -a
```

→ Noter que `ifconfig` n'est pas « fiable » (parce que l'alias n'a pas de nom)

→ Tuer un processus dans un VServer

```
# vserver monvs exec sleep 10000 &
# vps auxw | grep sleep
# kill <pid>
# vkill –c monvs <pid>
```

# « Entrer » dans le VServer

→ « Entrer » dans monvs à partir de l'hôte : crée un shell (bash par défaut) dans le contexte du VServer, après avoir fait le chroot vers son répertoire (plus quelques autres manips)

```
# vserver monvs enter
```

→ Vérifier que les autres processus (ceux de l'hôte) sont cachés

```
# ps auxw
USER       PID %CPU %MEM    VSZ   RSS COMMAND
root         1  0.2  0.5   1944   640 init [2]
root      2189  0.0  0.4   1632   572 /sbin/syslogd
root      2210  0.0  0.5   2200   756 /usr/sbin/cron
root      2251  7.5  0.0    120    36 login
root      2276  0.5  1.2   2736  1544 /bin/bash -login
root      2280  0.0  0.7   2228   904 ps auxw
```

→ Vérifier que les autres adresses réseaux sont cachées

```
# ifconfig –a
```

→ Et que le réseau marche

```
# ping www.renater.fr
PING www.renater.fr (193.49.159.10) 56(84) bytes of data.
64 bytes from 193.49.159.10: icmp_seq=1 ttl=128 time=48.5 ms
```

# Ajouter un service SSH

→ Installer SSH dans monvs et vérifier qu'il fonctionne

```
# apt-get install openssh-server
# ps auxw
```

⓵ Le service sshd ne s'est pas lancé correctement dans notre VServer !

Normal, car le serveur SSH de l'hôte fonctionne déjà, et il est en écoute sur toutes les adresses. Ce qui empêche sshd de démarrer dans le VServer. Il faut modifier la configuration sur l'hôte pour forcer sshd à n'écouter que sur l'adresse IP dédiée à l'hôte

→ Ajouter dans le fichier `/etc/ssh/sshd_config` (**sur l'hôte**) :
**`ListenAdress <ip.de.l.hote>`**

→ Relancer sshd sur l'hôte et vérifier qu'il ne rentrera plus en concurrence avec les VServers :

```
# /etc/init.d/ssh restart
```

# Ajouter un service SSH

```
# netstat -tpl
tcp […]192.168.44.128:ssh  *:*  LISTEN 1038/sshd
```

→ Essayer de lancer le service SSH dans le VServer

```
# /etc/init.d/ssh start
```

→ Vérifier qu'il fonctionne

```
# ps auxw
```

→ Depuis l'hôte, essayer de faire une connexion SSH vers le VServer : quel mot de passe ??

# Paramétrage initial du VServer

- Lors d'une installation avec debootstrap, il manque en effet des paramètres au VServer (comme le mot de passe du compte *root*)

→ Mettre un mot de passe à *root*

```
# passwd
```

→ Pour sélectionner la zone de temps, relancer la configuration du package Debian *tzdata*. Remarquer que le réglage de l'heure est une tâche du noyau, donc de l'hôte, mais la zone est propre à chaque VServer

```
# dpkg-reconfigure tzdata
```

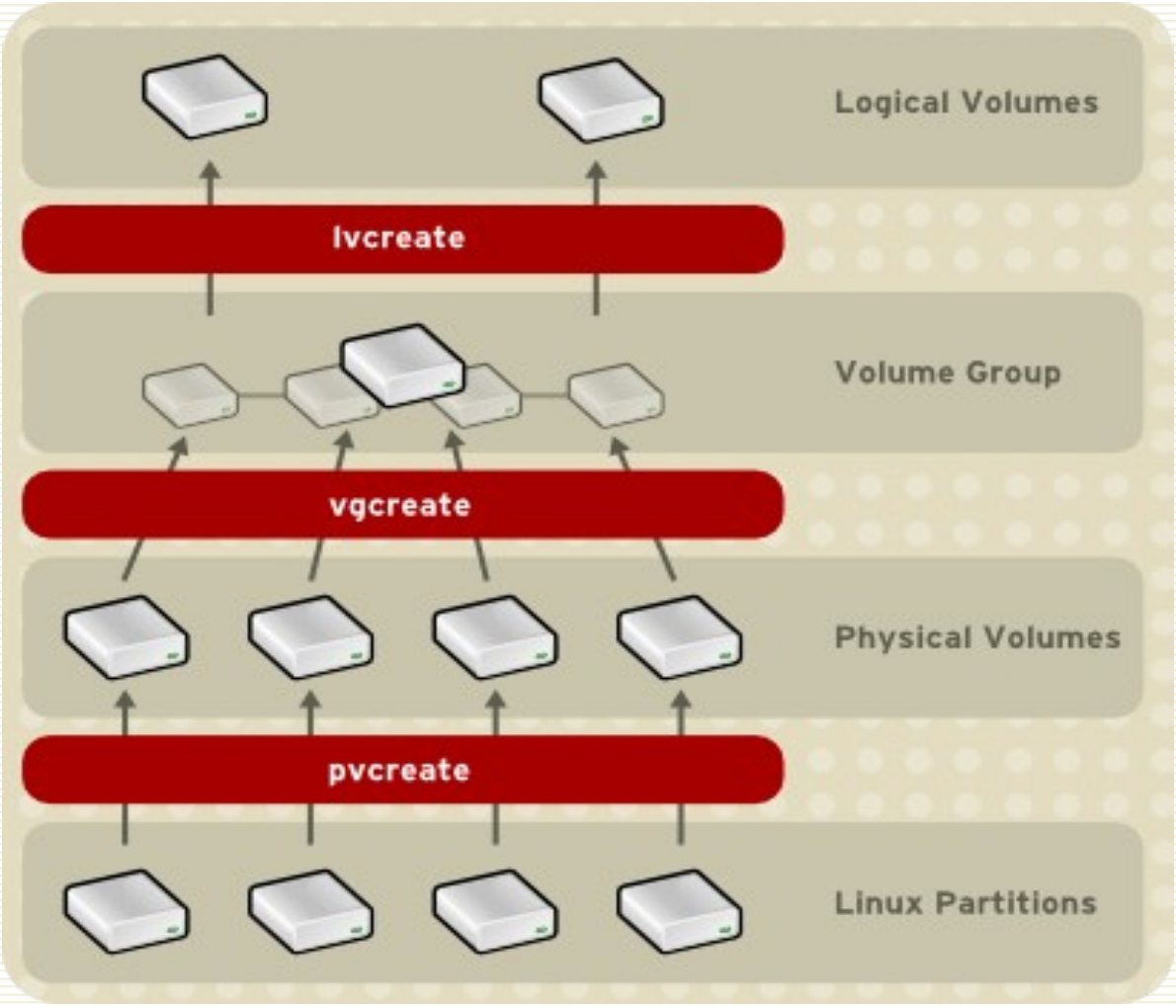
# Mettre le VServer dans un volume LVM (1)

- Notre installation a utilisé un répertoire pour l'espace disque (le chroot) du VServer. Ce répertoire est situé dans la partition `/` (sous `/var/lib/vservers`) de l'hôte
- Inconvénients :
  - Pour voir la taille occupée par l'hôte et par les VServers : la commande `df` ne suffit pas !
  - Pour limiter la taille d'un VServer ; sans limite, il pourrait remplir le disque de l'hôte (bloquer l'hôte, bloquer les autres VServers)
  - Si vous voulez mettre des quotas par utilisateur dans le VServer : ça ne marchera pas (bien), car les quotas systèmes sont fixés au niveau d'une partition, en fonction de l'UID attribué aux utilisateurs

# Mettre le VServer dans un volume LVM (2)

- Deux solutions :
  - Utiliser le « Disk Limit » par contexte. Nécessite un marquage des fichiers pour enregistrer le numéro de contexte (=de VServer) auxquels ils appartiennent. Le XID est inscrit dans les attributs UID/GID des fichiers, en utilisant les bits de poids fort
  - Utiliser des partitions différentes pour chaque VServer. Dans ce cas, mieux vaut utiliser LVM pour avoir de la souplesse (tailler les partitions au mieux, les retailler, etc.)
- Rappel sur LVM :
  - Les supports physiques (disques, partitions) sont appelés des *physical volume*
  - Les *physical volume* sont regroupés dans des *volume group* équivalents à des disques virtuels
  - Dernier niveau, on crée dans les *volume group* des *logical volume* équivalents à des partitions virtuelles
  - On peut alors formater le *logical volume* avec son système de fichiers favori
  - L'unité de base utilisé par LVM est un *extend* (équivalent au bloc)

# Schéma de LVM



# Mettre le VServer dans un volume LVM (3)

```
# pvcreate /dev/sda3
  Physical volume "/dev/sda3" successfully created
# vgcreate vserver /dev/sda3
  Volume group "vserver" successfully created
# lvcreate -n monvs -L 500M vserver
  Logical volume "monvs" created
# mke2fs -j -L monvs /dev/vserver/monvs
# vserver monvs stop
# cd /var/lib/vservers
# mv monvs monvs.1
# mkdir monvs
# mount /dev/vserver/monvs monvs
# mv monvs.1/* monvs/
# rmdir monvs.1/
# vserver monvs start
```

- Modifier `/etc/fstab` pour rendre le montage de la partition automatique

```
/dev/vserver/monvs  /var/lib/vservers/monvs  ext3 defaults   0   2
```

# Démarrage automatique du VServer

- Lorsqu'on reboote l'hôte, le VServer s'arrête et ne redémarre pas ! Comment le démarrer lors de l'amorçage de l'hôte ?
- Le paramétrage se fait à travers des fichiers textes, présents sous `/etc/vservers/<nom_du_vs>/`
  - Documenté sur http://www.nongnu.org/util-vserver/doc/conf/configuration.html
  - ⓘ Les fichiers n'existent pas forcément, il faudra les créer : attention aux erreurs de frappe !
  - Pour indiquer le niveau de démarrage automatique, on inscrit une étiquette dans le fichier `apps/init/mark`
  - Par défaut, seront démarrés (par le service `/etc/init.d/util-vserver`) ceux qui ont l'étiquette « **`default`** »

→ Rajouter l'étiquette « **`default`** » à monvs :

```
# echo default > /etc/vservers/monvs/apps/init/mark
```

→ Tester en rebootant l'hôte : monvs devrait démarrer tout seul

# TP – Plan

- Prise en main de l’hôte Debian
- Installation des composants
- Installer son premier VServer
- Dupliquer un VServer
- Pour aller plus loin …

# Dupliquer un VServer

- Pré-requis :
  - Avoir installé la distribution GNU/Linux Debian version «squeeze» sur un serveur
  - Avoir installé les composants Linux VServer
  - Avoir installé un VServer
- Objectif :
  - Apprendre à créer un nouveau VServer par recopie d’un VServer existant
  - Mieux connaître les fichiers et répertoires qui composent un VServer

# Dupliquer avec la commande vserver

- Le nouveau VServer s'appellera *nono*
- La commande `vserver … build` offre une méthode `rsync` qui permet de donner une source pour la construction du répertoire de chroot. Nous utiliserons cette méthode pour faire la recopie en local (rsync permet de faire une recopie à distance, pour recopier un vserver distant, ou pour recopier une machine physique)

→ Créer un volume logique dédié à nono, de 500 Mo

```
# lvcreate -n nono -L 500M vserver
# mke2fs -j -L nono /dev/vserver/nono
# cd /var/lib/vservers
# mkdir nono
# mount /dev/vserver/nono nono
```

→ Faire la recopie

```
# vserver nono build -m rsync \
    --context 200 --hostname nono.cume.fr \
    --interface nono=eth0:192.168.44.151/24 -- \
    --source /var/lib/vservers/monvs/
```

→ Démarrer nono et vérifier

```
# vserver nono start

# vserver-stat
# ip addr
# vserver nono enter
```

# Sauvegarder un VServer

- Rappels :
  - les paramètres d'un VServer sont stockés dans des fichiers, sous `/etc/vservers/<nom_du_vs>/`
  - Le répertoire du chroot est `/var/lib/vservers/<nom_du_vs>/` (particularité Debian pour respecter la hiérarchie standard FHS, `/vservers` à l'origine)
- Dans certains cas, il faut aussi sauver `/var/lib/vservers/.pkg/<nom_du_vs>/`
- Pour faire une recopie « manuelle » ou une sauvegarde, il faut recopier/sauver ces 3 répertoires
- Attention, si vous dupliquez un VServer « à la main », ou si vous descendez une sauvegarde sur un autre serveur :
  - Le répertoire de configuration contient des liens symboliques. Il faudra vérifier ces liens (`cache`, `run`, `vdir`)
  - Il faudra aussi vérifier qu'il n'y a pas de conflit dans la configuration ; regarder au moins les fichiers qui indiquent le nom du VServer (`name`), son nom d'hôte (`uts/nodename`), éventuellement son numéro de contexte (`context`) et son adresse IP (`interfaces/0/ip`)

# Un peu de pratique

- Installer un vserveur qui fera office de serveur dns
- Installer un vserveur qui fera office de serveur dhcp
- Installer un vserveur qui fera office de serveur LAMP

# TP – Plan

- Prise en main de l’hôte Debian
- Installation des composants
- Installer son premier VServer
- Dupliquer un VServer
- Pour aller plus loin …

# Agrandir `/tmp` dans un VServer

- `/tmp` est un système de fichiers tmpfs, avec une taille de 16 Mo
  - Le système de fichiers *tmpfs* est un système de fichiers en mémoire vive, utilisé pour des données temporaires
  - Taille limitée
  - Pas de persistence à travers les reboot
- Pour augmenter la taille pour un seul VServer, modifier son fichier de paramétrage `/etc/vservers/<vs>/fstab`. Ce fichier est lu au démarrage du VServer, il faudra donc redémarrer le VServer pour qu'il soit pris en compte
  - Faites la modification sur nono par exemple et vérifier le résultat
- Pour augmenter ce paramètre pour tous les nouveaux VServer construit avec « `vserver … build` », le fichier modèle est `/usr/lib/util-vserver/defaults/fstab` . Mais il ne faut pas le modifier, car il sera écrasé lors des mises à jours du paquet util-vserver. Vous devez le recopier sous `/etc/vservers/.defaults/` et modifier cette copie.
- Pour rendre `/tmp` persistant, on peut se passer de tmpfs et utiliser un `/tmp` « classique ». Pour cela, vous pouvez simplement mettre la ligne de tmpfs en commentaire dans fstab

# Agrandir /tmp dans un VServer (solution ...)

→ Augmenter le /tmp de nono à 32 Mo en modifiant le fichier /etc/vservers/nono/fstab

→ Redémarrer nono

```
# vserver nono start
```

→ Vérifier que l'espace de /tmp s'est effectivement accru

```
# vserver nono enter
# df
Filesystem    1K-blocks      Used Available Use% Mounted on
/dev/hdv1        495844    141319    328925  31% /
none              32768         0     32768   0% /tmp
```

# Changer l'adresse IP et le nom de l'alias sur interface réseau d'un VServer

- L'adresse IP est dans le fichier `interfaces/0/ip`
- Avec la commande `vserver … build` utilisée, l'alias sur eth0 n'a pas de nom
  - On peut indiquer un nom, qui sera inscrit dans le fichier `interfaces/0/name`
  - Pour indiquer un nom d'alias lors du build du VServer, on aurait pu utiliser la syntaxe
    `--interface` **`monvs=`**`eth0:192.168.44.151/24`

# Changer l'adresse IP et le nom de l'alias sur interface réseau d'un VServer (2)

→ Arrêter monvs

```
# vserver monvs stop
```

→ Changer l'IP de monvs

```
# cd /etc/vservers/monvs
# echo 192.168.44.160 > interfaces/0/ip
```

→ Donner un nom à son alias sur eth0

```
# echo monvs > interfaces/0/name
```

→ Démarrer monvs et vérifier

```
# vserver monvs start
# ip addr ls
```

→ Noter que la commande ifconfig montre cette fois-ci l'alias, parce qu'il a un nom

```
# ifconfig –a
```

# Renommer un VServer

- Le nom du VServer à strictement parler correspond au nom du répertoire `/etc/vserver/`*`nom_du_vs`*
- Le nom associé au contexte utilisé par le VServer est stocké dans le fichier `name`

→ Renommer *nono* (le petit robot) en *ulysse* :

```
# vserver nono stop
# cd /etc/vservers
# mv nono ulysse
# cd ulysse
# echo ulysse > name
```

→ Vérifier que le changement de nom est effectif

```
# vserver ulysse start
# vserver-stat
```

# Renommer un VServer (2)

- Problème : nous avons une configuration « bancale », source d'erreurs potentielles, car il y a d'autres noms à modifier pour rester cohérent !

→ Changer le nom de l'alias sur eth0 et le nom d'hôte :

```
# vserver ulysse stop
# echo ulysse > interfaces/0/name
# echo ulysse.cume.fr > uts/nodename
```

→ Changer le nom du *logical volume* LVM et le nom du point de montage ; il faut changer des liens symboliques du répertoire de configuration

```
# cd /var/lib/vservers
# umount nono
# mv nono ulysse
# lvrename /dev/vserver/nono ulysse
# mount /dev/vserver/ulysse ulysse
# cd /etc/vservers/ulysse
# ls –l
# rm vdir   && ln -s /etc/vservers/.defaults/vdirbase/ulysse  vdir
# rm cache  && ln -s /etc/vservers/.defaults/cachebase/ulysse cache
# rm run    && ln -s /var/run/vservers/ulysse run
```

→ Ne pas oublier de modifier /etc/fstab de l'hôte !

# Spécifier un miroir Debian lors du build

- Lors du build des VServer Debian, le miroir utilisé fut celui de Debian, alors que nous avons des miroirs plus proches
- Le fichier des sources Debian dans les VServer pointe aussi sur le miroir Debian

```
# cat /etc/apt/sources.list
deb http://ftp.debian.org/debian etch main
```

→ Modifier le miroir par défaut :

```
# cd /etc/vservers
# mkdir -p .defaults/apps/debootstrap/
# cd .defaults/apps/debootstrap/
# echo http://ftp.lip6.fr/pub/linux/distributions/debian/ > mirror
```

# Limiter la mémoire consommée

- La cohabitation des VServers (entre eux et avec l'hôte) se veut « coopérative », comme des processus UNIX classiques
- En cas de manque de mémoire, le « Out-of-Memory (OOM) Killer » du noyau fonctionnera (comment sélectionnera-t'il un processus ?)
- Mémoire résidente (rss) : nombre de pages mémoires utilisées en RAM
- Mémoire virtuelle (espace d'adressage, as) : nombre total de pages mémoires allouée
- Mémoire partagée : deux processus peuvent partager des pages en mémoires. Mécanisme utilisé par les librairies partagées (.so sous Linux, .dll de Windows), afin de ne charger qu'une seule fois en mémoire une librairie utilisée par plusieurs processus
- Attention : la comptabilisation de la mémoire virtuelle pour un VServer est « trompeuse », car une page de mémoire partagée utilisée par deux processus sera comptée deux fois
- Une page mémoire = 4ko (sur Linux i386/em64t)
- On peut limiter la mémoire consommée par un VServer (http://linux-vserver.org/Memory_Limits)

# Limiter la mémoire consommée (2)

→ Limiter la taille en RAM à 5000*4ko = 20Mo

```
# vserver ulysse stop
# cd /etc/vservers/ulysse
# mkdir rlimits
# echo 5000 > rlimits/rss.hard
```

→ « Virtualiser » l'affichage de la mémoire disponible

```
# echo virt_mem > flags
```

→ Démarrer ulysse

```
# vserver ulysse start
```

→ Afficher l'utilisation actuelle, le max atteint, la limite hard, le nombre de fois où le max a été atteint

```
# cat /proc/virtual/200/limit
RSS:        453         1248         5000        0
```

# Limiter la mémoire consommée (3)

→ Afficher la mémoire vue dans ulysse, et noter que la vue est effectivement « virtualisées » : la mémoire (vive) vue est la valeur fixée par RSS max

```
# vserver ulysse enter
# top
# cat /proc/meminfo
MemTotal:        20000 kB
```

→ Tester à l'aide d'un programme en C (`malloc` alloue puis remplit un tableau de X Mo). On demande ici une taille de 30 Mo supérieure à la limite fixée :

```
# wget http://scipc-xm.unilim.fr/malloc
# chmod +x malloc
# ./malloc 30
Allocation du tableau (30 Mo) : OK.
Remplissage du tableau (Mo) : 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27 28 29.
```

ⓘ Ca marche ! Le noyau Debian a une version trop ancienne de VServer !

→ Sur une machine avec VServer 2.2, on obtient le message :

```
# ./malloc 30
Allocation du tableau (30 Mo) : OK.
Killed
```

→ Le noyau autorise l'allocation du tableau, mais le programme est tué (par le OOM killer) lorsque la consommation réelle dépasse la limite

# Limiter la mémoire consommée (4)

→ Nous limitons maintenant la mémoire **virtuelle** à 8000*4ko = 32Mo

```
# vserver ulysse stop
# cd /etc/vservers/ulysse
# echo 8000 > rlimits/as.hard

# vserver ulysse start
# vserver ulysse enter
# top
# cat /proc/meminfo
MemTotal:        20000 kB
SwapTotal:       12000 kB
```

→ La taille du fichier d'échange (swap) = mémoire virtuelle max (AS) – mémoire vive (RSS)

→ On teste à nouveau l'allocation mémoire

```
# ./malloc 30
Allocation du tableau (30 Mo) : KO.
Cannot allocate memory
```

→ Le noyau refuse l'allocation, le programme peut s'arrêter « proprement »

⓿ Limite : l'espace d'adressage ne représente pas grand-chose de réel …

# Limiter la CPU consommée

- Linux-VServer utilise un algorithme « seau de jeton » pour limiter la CPU consommée par un VServer
  - A chaque « tick » d'horloge, un processus en cours d'exécution consomme un jeton du seau
  - Le seau se remplit de R jetons (*fill rate*) tous les T ticks
  - ⇨ Le % de CPU que le VServer pourra consommer sera de R/T*100
  - Le seau a une capacité maximale
  - Si le seau se vide, il devra se remplir jusqu'à un certains seuil avant que les processus puissent à nouveau piocher dedans
  - ⇨ Offre une capacité de « pic » de CPU plus ou moins long pour un VServer qui aura économisé des jetons
- Garantie : si la somme des Ri/Ti ≤ 1 (pour tous les VServer i), alors on obtient une **garantie** (réservation) de ressource CPU. On est certains qu'ils disposeront de cette capacité
- Partage équitable : si certains VServers ne travaillent pas, on perd de la CPU. Le *partage équitable* (*fair share*) permet de partager le temps CPU inutilisé : à chaque VServer, on associe un deuxième taux R'/T'. Un VServer ayant un taux R'/T' deux fois plus important qu'un autre récupérera deux fois des crédits CPU inutilisés
  - ⓘ Le *fair share* est disponible en 2.2, pas en version 2.0 …

# Limiter la CPU consommée (2)

→ Nous allons installer un programme de stress CPU dans ulysse

```
# vserver ulysse enter
# apt-get install cpuburn
# burnMMX
```

→ Vérifier l'utilisation CPU depuis l'hôte

```
# vtop
PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 5690 root      25   0 65636   84    4 R 99.9  0.1   0:32.29 burnMMX
```

→ Limiter la consommation d'ulysse à 25% (1/4) de la CPU

```
# vserver ulysse stop
# cd /etc/vservers/ulysse
# mkdir sched
# echo 1 > sched/fill-rate
# echo 4 > sched/interval
```

→ Activer le scheduler CPU (sinon, les paramètres ne sont pas pris en compte)

```
# echo sched_hard >> flags
```

→ Redémarrer ulysse

```
# vserver ulysse start
```

# Limiter la CPU consommée (3)

→ Vérifier les paramètres de l'ordonnanceur de priorité. Pour l'instant, il ne fait rien (ou presque), les lignes Token et TokenMax indiquent que le seau est rempli

```
# cat /proc/virtual/200/sched
Token:                125
FillRate:               1
Interval:               4
TokensMin:             15
TokensMax:            125
```

→ Relancer le programme de stress CPU

```
# burnMMX
```

→ Vérifier l'utilisation CPU depuis l'hôte

```
# vtop
top - 11:07:53 up  6:27,  2 users,  load average: 0.39, 0.39, 0.21
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
19725 root      25   0 65632   80    4 H 25.5  0.1   0:11.31 burnMMX
```

→ Noter que top dans le VServer, ou vtop dans l'hôte, montrent tous les deux les mêmes infos

→ Mettre une priorité différente (1/2) pour le VServer « monvs » et lancer le programme de charge sur les deux VServers

# Virtualiser l'affichage de la CPU consommée

→ Activer l'affichage virtualisée de la charge dans le VServer, en fonction de ce qu'il utilise et de ce qui lui est alloué

```
# cd /etc/vservers/ulysse
# echo virt_load >> flags
```

→ Relancer ulysse et vérifier que l'affichage de la charge est différent avec l'hôte (avec la commande `w`) :

```
# vserver ulysse restart

# vserver ulysse enter
# burnMMX &
# w
11:12:12 up 2 min,  0 users,  load average: 1.05, 0.41, 0.15

# w
11:12:14 up 2 min,  2 users,  load average: 0.51, 0.36, 0.13
```

→ L'affichage de la charge est « personnalisé » dans le VServer

# Virtualiser l'affichage de l'uptime

- Dans l'utilisation de top qui précède, vous avez peut-être vu que l'affichage de l'uptime est identique dans l'hôte et dans le VServer
- On peut « virtualiser » l'affichage de l'uptime dans un guest. Pour cela, il faut positionner un *flag* sur le VServer
- Où met-on des flags ? Voir la « flower page »
  http://www.nongnu.org/util-vserver/doc/conf/configuration.htm
  ou http://linux-vserver.org/util-vserver:Documentation
- Chercher le nom du flag : voir
  http://linux-vserver.org/Capabilities_and_Flags
- Vérifier que ça marche !

# Nettoyer les namespace

- Un VServer possède son propre « namespace » au niveau de la couche VFS de Linux. Un namespace est une vue des systèmes de fichiers montés
- Lorsqu'un VServer démarre, son namespace hérite de celui de l'hôte, puis il y a un « nettoyage » afin qu'il ne voit pas les systèmes de fichiers montés sur l'hôte (ils sont démontés dans le namespace du VServer)

- La commande `mount` n'est pas pertinente ; pour voir les systèmes de fichiers montés, regarder dans `/proc/mounts` sur l'hôte !

# Nettoyer les namespace (2)

→ Regarder les montages sur l'hôte

```
# grep vserver /proc/mounts
/dev/vserver/monvs /var/lib/vservers/monvs ext3
/dev/vserver/ulysse /var/lib/vservers/ulysse ext3
```

→ Noter que dans l'hôte, on ne voit pas les montages de /proc, /tmp etc. des vservers car ce montage est fait uniquement dans leur propre namespace

→ Regarder les montages dans un autre contexte (du point de vue d'ulysse)

```
# vnamespace –e 200 grep vserver /proc/mounts
/dev/vserver/ulysse /var/lib/vservers/ulysse ext3
none /var/lib/vservers/ulysse/proc proc
none /var/lib/vservers/ulysse/tmp tmpfs
none /var/lib/vservers/ulysse/dev/pts devpts
/dev/vserver/ulysse / ext3
```

→ Noter que la commande vnamespace permet de rentrer dans le namespace d'un VServer, en indiquant son numéro de contexte. Ici, on voit que dans le namespace du vserver ulysse, « monvs » n'est pas monté

# Nettoyer les namespace (2)

→ Désactiver cette fonction de nettoyage pour ulysse :

```
# vserver ulysse stop
# cd /etc/vservers/ulysse
# touch nonamespace-cleanup
# vserver ulysse start
# vnamespace –e 200 grep vserver /proc/mounts
/dev/vserver/monvs /var/lib/vservers/monvs ext3
[…]
```

→ Noter que maintenant on voit le montage de l'autre VServer (monvs)

→ La désactivation de cette fonction peut être faite de façon globale (`etc/vservers/.defaults/nonamespace-cleanup`)

→ On peut aussi désactiver l'usage des namespace

→ En fonctionnement « normal », il est préférable de ne pas toucher à ça !

# Installer une autre distribution – Avec yum

- Il est possible d'installer une distribution différente de celle de l'hôte dans un VServer. En général, les services qu'on souhaite installer (serveur web, base de données, serveur SSH, DNS, mail, ...) ne sont pas dépendants du noyau utilisé. Ils fonctionneront donc sans problème dans le VServer
- La méthode « yum » permet d'installer des distributions RPM (équivalent Fedora/Red Hat de « debootstrap »)
- Les « distributions » sont décrites sous `/usr/lib/util-vserver/distributions/`
- Liste un peu ancienne ...
  - On peut en rajouter sous `/etc/vservers/.distributions/`
  - Contient la configuration de yum ou apt (serveurs à utiliser, ...)
  - Contient la liste des paquets à installer (dossier `pkg/`)
  - Un script à exécuter avant et après le build (initpre et initpost)

# Installer une autre distribution – Avec yum (2)

→ Construire un VServer nommé « *fc6* » en utilisant la méthode *yum* et la ditribution « fc6 » :

```
# apt-get install yum sqlite python-sqlite
# vserver fc6 build -m yum \
    --context 2006 --hostname fc6.cume.fr \
    --interface fc6=eth0:192.68.44.156 \
    -- -d fc6
```

→ Si yum n'arrive pas à télécharger les paquets depuis les sites déclarés par défaut, on personnalise la distribution

```
# cd /etc/vservers/.distributions/fc6/
# cp  –a /usr/lib/util-vserver/distributions/fc6/yum.repos.d .
# cd yum.repos.d/
# vi fedora-core.repo
baseurl=ftp://ftp.lip6.fr/pub/linux/distributions/fedora/6/$basearch/os/
# rm fedora-development.repo fedora-extras-development.repo \
    fedora-legacy.repo fedora-updates-testing.repo
```

→ Démarrer le VServer

```
# vserver fc6 start
```

# Installer une autre distribution – Par archive

- Autre méthode générique :
  - Installer la distribution sur une machine témoin, dans une version minimaliste (la machine témoin peut être une machine virtuelle VMware, Qemu !)
  - Faire une archive tar de la machine
  - Déployer avec la méthode de build « template », qui se sert d'une archive pour créer le répertoire du chroot
- Avantages :
  - L'installation se fait de manière « classique », avec les CD de la distribution
  - Le template créé servira plusieurs fois
- Inconvénients :
  - Un travail de préparation
  - Installe dans le VServer de service inutile qui essayent d'interagir avec le noyau, ce qui est reservé aux processus de l'hôte. Ces services généreront des messages d'erreur au démarrage et à l'arrêt du VServer (mais sans dommage).
    **Exemples de services qui n'ont pas de sens au sein d'un VServer** : modification des fontes de la console, ACPI, réglage de l'heure système, configuration de la couche réseau (nom d'hôte, adresse IP, tables de routage), découverte des modifications du matériel, montages des systèmes de fichiers, etc.

# Installer une autre distribution – Par archive (2)

→ Télécharger le template http://scipc-xm.unilim.fr/mandriva.tar.gz, qui est une archive de la partition racine (/) d'une Mandriva 2007.1 installée dans une machine Qemu (temps nécessaire : 1 heure)

→ Construire un VServer nommé *mandriva* en utilisant la méthode *template* :

```
# vserver mandriva build -m template \
    --context 2007 --hostname mandriva \
    --interface mandriva=eth0:192.68.44.157 \
    -- -t mandriva.tar.gz
```

→ Démarrer mandriva

```
# vserver mandriva start
```

Note sur l'utilisation de Qemu

→ Installation de Qemu
`apt-get install qemu kqemu-common kqemu-modules-2.6-<arch>`

→ Création d'un fichier vide pour le disque dur
`dd if=/dev/zero of=/tmp/disk.raw bs=1 count=1 seek=3G`

→ Démarrage de Qemu
`qemu -hda /tmp/disk.raw –cdrom dvd-mandrake.iso -boot d`

→ Installation de Mandrake dans Qemu, en version « minimaliste »

→ Création de l'archive

# Virtualiser un serveur existant

- « P2V » (*Physical to Virtual*)
- A peu près identique à ce que nous avons fait pour construire un VServer d'un autre type que Debian de façon générique
- Vous pouvez utiliser une archive tar
- Vous pouvez aussi utiliser la méthode « rsync » pour recopier la machine existante
- Si vous utilisez une autre méthode, prenez garde à ne pas recopier le contenu du répertoire `/dev`
  - `/dev` ne doit être peuplé que de quelques fichiers périphériques bien précis
  - Sinon, vous ouvrez des portes d'accès depuis le VServer vers l'hôte
  - Pour créer `/dev`, vous pouvez le recopier depuis un autre VServer, ou utiliser la méthode de build « *skeleton* » qui se contente de créer le répertoire de configuration, puis de peupler `/dev` correctement
- Pourquoi faire ?
  - Faire un test de migration applicatif, mise à jour
  - Dupliquer une installation existante
  - Se débarrasser d'une vieille « bécane » (partie matérielle)
  - …

# Gérer les paquets des VServers depuis l'hôte

- On peut gérer depuis l'hôte les paquets des VServers (s'ils sont allumés)
- Pratique pour administrer d'un bloc plusieurs VServers

→ Rafraîchir, sur tous les VServes, la liste des paquets Debian disponibles

```
# vapt-get --all -- update
```

→ Installer le paquet « file » sur monvs

```
# vapt-get monvs -- install file
```

# Séquence de démarrage

- La séquence de démarrage d'un VServer s'affiche en détail avec l'option « `--debug` »
- Beaucoup d'actions et de possibilités, la description qui suit simplifie certains détails ou séquences

- Création d'un nouveau namespace pour les systèmes de fichiers montés. Un namespace ne porte pas de nom, tous les processus suivants hériteront de cette propriéte et seront dans ce namespace
- Exécution des scripts `scripts/initialize, scripts/initialize.d/*` , puis de `.defaults/scripts/initialize` et `.defaults/scripts/initialize.d/*`
- Montage de la racine du systèmes de fichiers (`/`) du VServer en utilisant le fichier `fstab` (par défaut, il n'y a pas d'entrée pour /)
- Exécution des scripts `prepre-start`
- Ajout de l'interface réseau dédiée au VServer sur l'interface de l'hôte
- Montage des autres systèmes de fichiers de `fstab` (autre que `/`)
- Exécution des scripts `pre-start`

# Séquence de démarrage (2)

- Toute la suite s'exécute avec la valeur « nice » spécifiée dans le fichier `nice`
- Toute la suite s'exécute limité à l'adresse IP du VServer
- Crée un nouveau contexte avec le numéro spécifié dans le fichier `context`, et tout le reste sera fait dans ce contexte
- Enregistre le namespace courant comme le namespace du contexte
- Positionne les limites d'utilisation de ressources (mémoire, nombre de fichiers ouverts, …) en fonction des données spécifiées dans le répertoire `rlimits/`
- Positionne les limites d'utilisation CPU pour le scheduler en fonction des données du répertoire `sched/`
- Positionne les affichages virtualisés *utsname* (nom d'hôte, version d'OS, …), en fonction des données du répertoire `uts/`
- Positionne les attributs (flag) du VServer (virt_mem, …), en fonction du contenu du fichier `flags`
- Lance la commande « `/etc/init.d/rc 3` » après avoir fait un chroot dans le répertoire du VServer. Le niveau de runlevel utilisé, la commande utilisée, est fonction du paramétrage sous `apps/init`

# Limiter l'espace disque d'un VServer sans LVM

- Si on n'utilise pas LVM : on peut marquer les fichiers avec le XID (numéro de contexte)
- Utile pour mettre un limite d'espace disque (*disk limit*) pour le VServer
- Doc : http://oldwiki.linux-vserver.org/Disk+Limits
- FS à monter avec l'option « tagxid » (VS 2.0) ou « tag » (VS 2.2)
  - Impossible d'utiliser l'option « -o remount » pour ajouter cette option !
  - Très déconseillé d'essayer de le mettre sur la partition /
  - ➲ A mettre par ex. sur une partition dédiée à tous les VServers (Cf http://www.paul.sladen.org/vserver/archives/200602/0020.html)

→ Créer une partition pour le test

```
# lvcreate --name testxid --size 500M vservers
# mke2fs –j /dev/vservers/testxid
```

→ Monter la partition avec l'option « tagxid »

```
# mkdir /var/lib/vservers/testxid
# mount –o tagxid /dev/vservers/testxid /var/lib/vservers/testxid
```

→ Créer un nouveau VServer par recopie

```
# vserver testxid build -m rsync --context 300 --hostname testxid.cume.fr \
    --interface eth0:192.168.44.153/24 -- --source /var/lib/vservers/monvs/
```

→ Avant mis en place des limites, vérifier que le VServer voit tout l'espace du disque

```
# vserver testxid start
# vserver testxid enter
# df
```

→ Arrêter le VServer et modifier le XID des fichiers du VServer :

```
# vserver testxid stop
# chxid -R -c 300 /var/lib/vservers/testxid
```

# Limiter l'espace disque d'un VServer sans LVM (2)

→ Mettre les limites, en créant le répertoire `dlimits/0/` et y mettre les fichiers directory (répertoire où appliquer les limites), inodes_total (nb d'inodes), reserved (% d'espace pour root), space_total (**en Ko**)

```
# cd /etc/vservers/testxid/
# echo /var/lib/vservers/testxid > dlimits/0/directory
# echo $((200 * 1024)) > dlimits/0/space_total
# echo 100000 > dlimits/0/inodes_total
# echo 5 > dlimits/0/reserved
```

→ Démarrer le VServer et contrôler les limites avec les commandes `vdlimit` et `vdu`

```
# vserver testxid start
# vdlimit --xid testxid /var/lib/vservers/testxid
300 /var/lib/vservers/testxid
space_used=154602
space_total=204800
inodes_used=7834
inodes_total=100000
reserved=5

# vdu --xid 300 --space /var/lib/vservers/testxid
/var/lib/vservers/testxid 154603
```

→ Et vérifier la limite à l'intérieur du VServer

```
# vserver testxid enter
# df
```

→ Si vous voulez supprimier les limites, il faut effacer le répertoire `dlimits/` et exécuter la commande

```
# vdlimit --xid testxid --remove /var/lib/vservers/testxid
```

# Rebooter un VServer depuis le VServer

- La commande « `reboot -f` » permet de rebooter le VServer **depuis** le VServer

- Par défaut, la commande « `reboot` » seule ne marche pas, l'option « `-f` » est obligatoire

- La commande de reboot est interceptée par un « intermédiaire » (*helper*) côté hôte, qui va faire (par défaut) un « `vserver <vs> restart` »
- Pour aller plus loin :
  - Le programme helper est « /sbin/vshelper » (déclaré / modifiable dans `/proc/sys/kern/vshelper`)
  - La commande à exécuter pour un reboot est modifiable. Voir http://www.nongnu.org/util-vserver/doc/conf/configuration.html#vshelper-action

# Un mot sur les messages d'erreur à l'arrêt des VServer

- Au sein d'un VServer, *root* ne peut pas tout faire
- Les scripts d'arrêt usuels de Linux font des opérations interdites au sein d'un VServer :
  - Démonter des systèmes de fichiers
  - Enregistrer l'heure dans l'horloge système
  - Déconfiguration du réseau
  - …
- Les messages d'erreur ne sont pas bloquants, seulement gênant
- Pour les éviter, il faut changer la séquence d'arrêt et **supprimer** ces opérations non permises
- Les versions plus récentes du paquet *util-vserver* prennent en charge le « nettoyage » du VServer à la fin de sa construction. Voir le script « `initpost` » sous `/usr/lib/util-vserver/distributions/<distrib>/`

# Quelques pièges « classiques »

- Modifier la configur ation du réseau sans avoir arrêté au préalable le VServer : le « stop » n'enlèvera peut-être pas l'adresse réseau, et le « start » suivant n'appliquera donc pas les modifications !
- Le nom de l'alias sur l'interface réseau ne doit pas être trop long – longueur max du nom complet de l'interface (y compris « eth0: ») = 15 caractères
- Oublier de monter le répertoire du chroot du VServer

# Ce qui n’a pas été dit ou fait …

- Illustrer indépendamment les éléments composants les VS (contexte, namespace, isolation réseau, flags, capacités, etc.)
- Faire des modifications « à chaud » sur un VS, sans le redémarrer
- Cpuset (affinité, fixer un VServer sur un processeur quand l’hôte est SMP) : intégré à la configuration des VServer
- Comment recompiler soi-même un noyau VServer
- Modification des capacités (rarement nécessaire, et dangereux en terme de sécurité)
- Style d’init (plain style vs. fakeinit)
- Messages d’arrêts « propres »

# Unification

- Utilise un VServer de « référence » pour économiser de l’espace disque :
  - Les autres VServers sont sur la même partition
  - Les fichiers des autres VServers sont des liens UNIX en dur. Le contenu du fichier n’est donc pas dupliqué
  - VServer implémente une technique de « copy-on-write » qui permet quand même de modifier les fichiers au sein d’un VServer sans impacter les autres
- Nécessite VServer 2.2

# Installer une version plus récente de VServer

- Sous GNU/Linux Debian version *etch*, on peut installer des logiciels dits « backportés », c-à-d des versions plus récentes, initialement non intégrées à *etch*
- Pour Linux VServer, il existe un noyau basé sur Linux 2.6.22 et VServer 2.2, ainsi que les commandes utilisateurs plus récentes :
  - http://packages.debian.org/search?keywords=linux-image-2.6-vs
  - http://packages.debian.org/search?keywords=util-vserver&searc

# Héberger des VServers dans des VLANs différents

- Dans nos TP, nous avons utilisé pour les VServers des adresses IP du même réseau que l'hôte
- Mais on peut être amené à héberger des VServer dédiés à des équipes différentes, à placer sur des réseaux différents
- Solution :
  - Utiliser des VLANs taggés pour amener plusieurs réseaux à travers un seul lien réseau
  - Créer une interface sur chacun des VLAN
  - Écrire plusieurs tables de routage (Linux supporte 256 tables de routages différentes)
  - Écrire des règles pour sélectionner la table de routage en fonction de l'adresse IP source

- Dans l'exemple suivant, nous aurons :
  - Deux VLAN, d'ID 7 et 8
  - Réseaux IP 10.7.X.Y/16 et 10.8.X.Y/16
  - Passerelles 10.7.0.1 et 10.8.0.1
  - Deux VServers vs_7 et vs_8, avec des IP 10.7.0.2 et 10.8.0.2

- Toute la configuration se fait sur l'hôte
- Ne pas oublier la configuration de l'hôte (non détaillé ici)

# Héberger des VServers dans des VLANs différents (2)

- La commande `vconfig` crée des interfaces sur les VLAN. Les interfaces porteront comme nom <*device*>.<*vlan_id*> :

```
# vconfig add eth0 7
# vconfig add eth0 8
# ifconfig eth0.7 up
# ifconfig eth0.8 up
```

- On crée des tables de routage différentes pour chacun des réseaux :

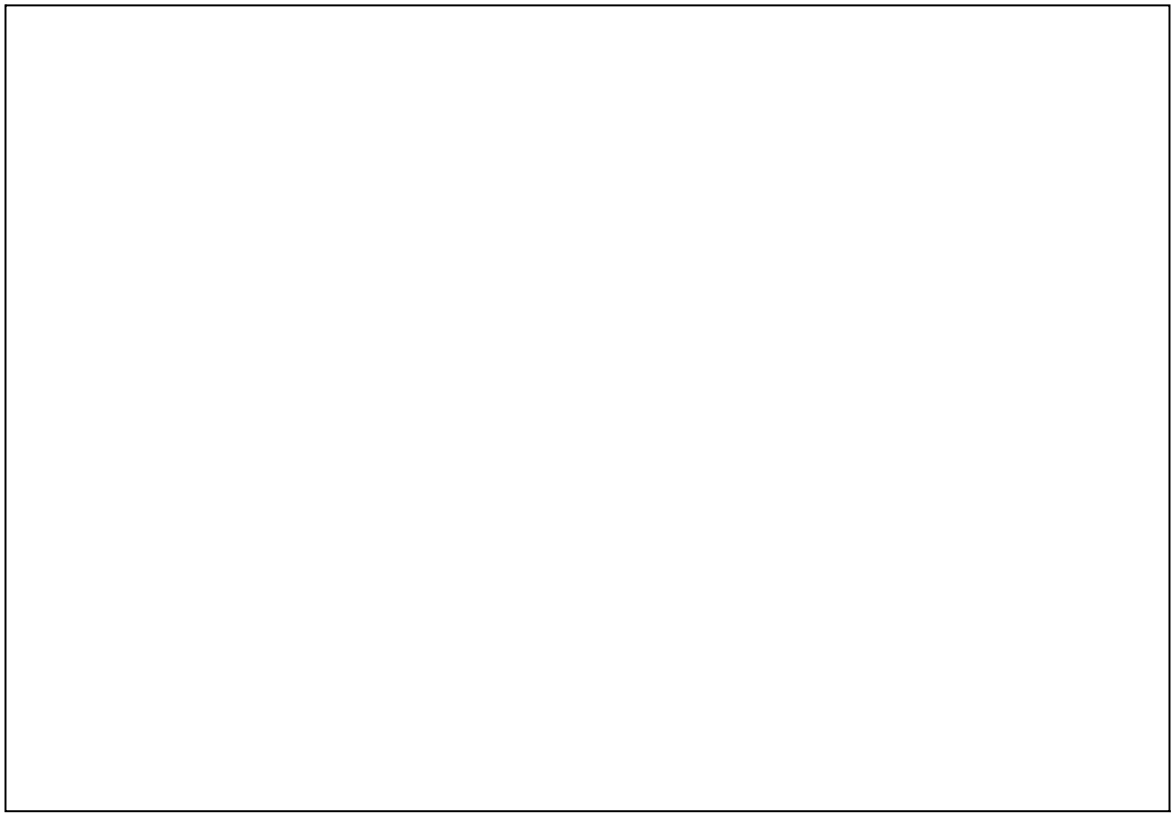
```
# ip route add 10.7.0.0/16 dev eth0.7 table 7
# ip route add default via 10.7.0.1   table 7
# ip route add 10.8.0.0/16 dev eth0.8 table 8
# ip route add default via 10.8.0.1   table 8
```

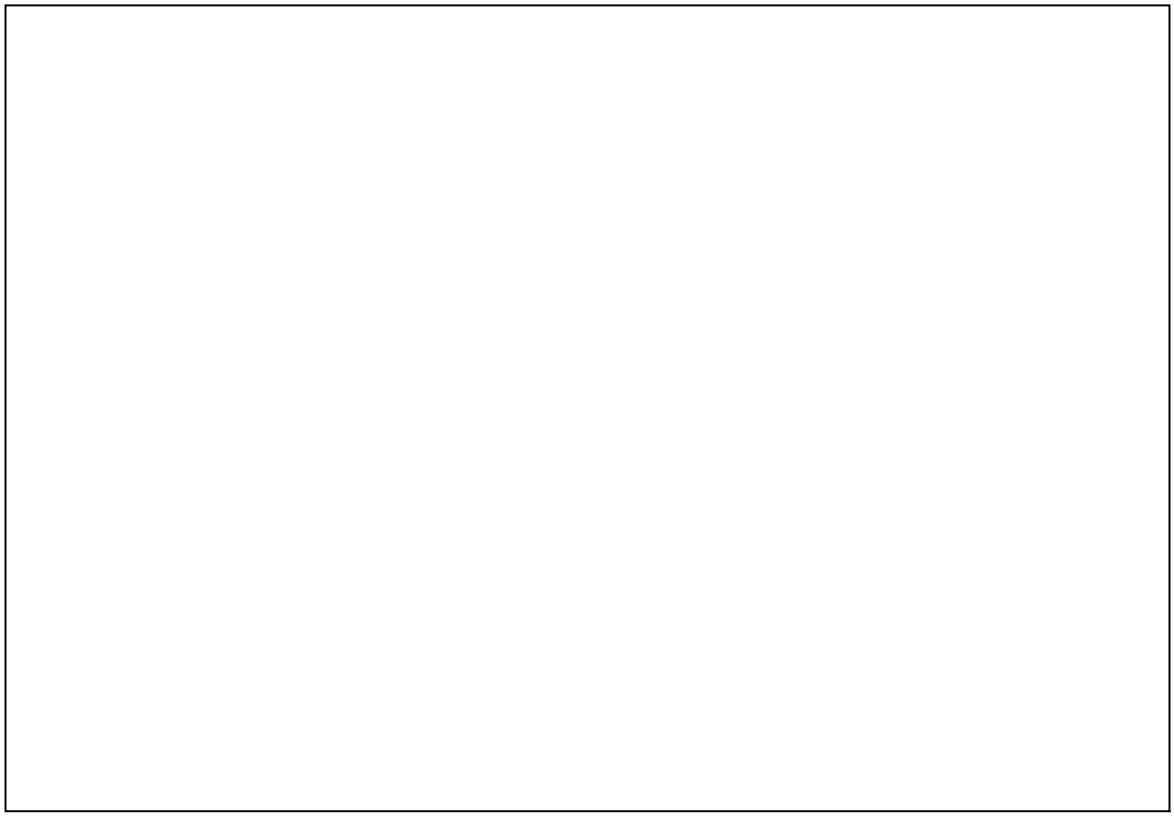
- On crée des règles pour sélectionner la table de routage en fonction de l'adresse IP source

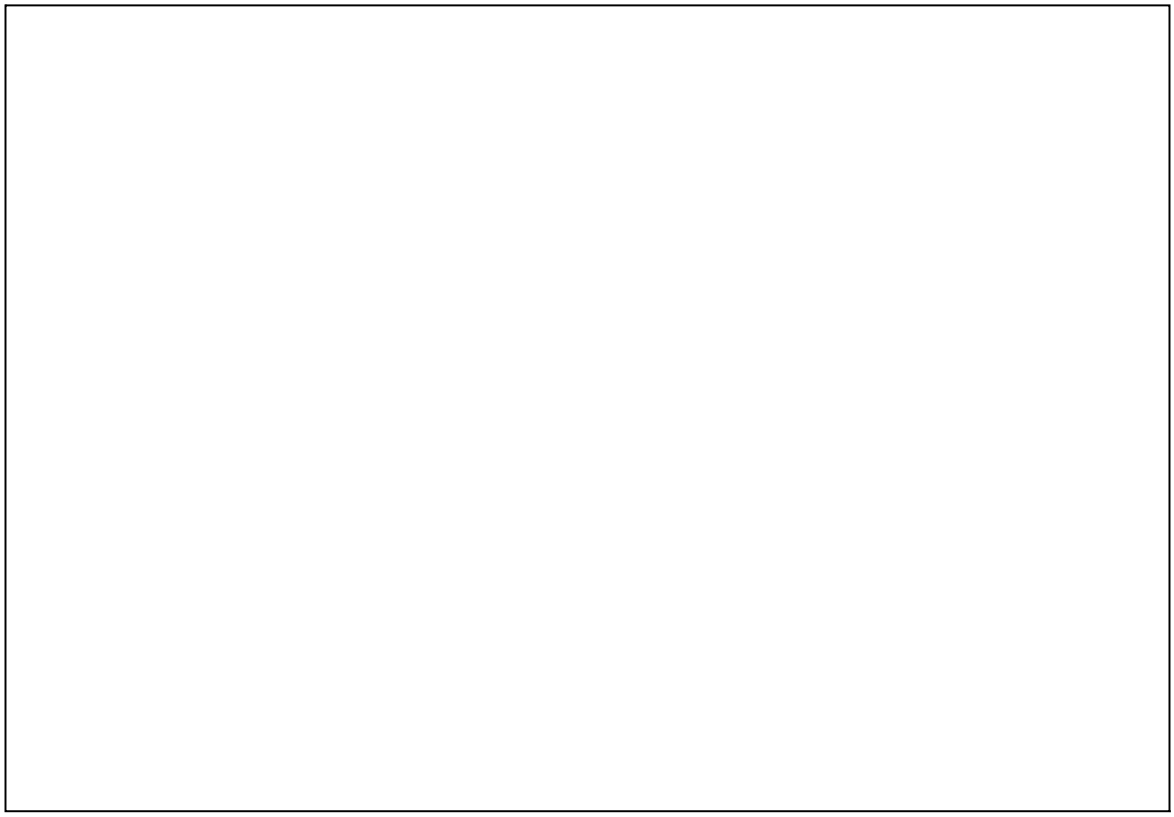
```
# ip rule add from 10.7.0.0/16 table 7
# ip rule add from 10.8.0.0/16 table 8
```

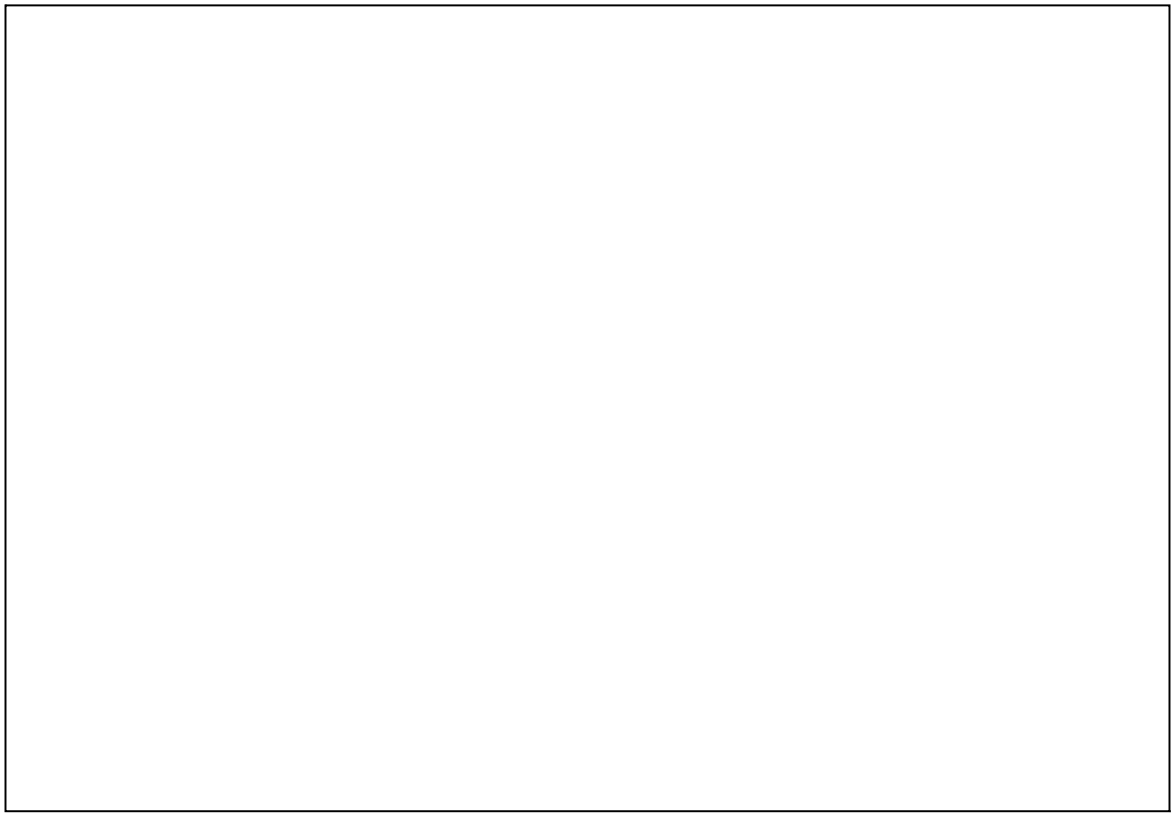
- On place les VServers sur ces réseaux :

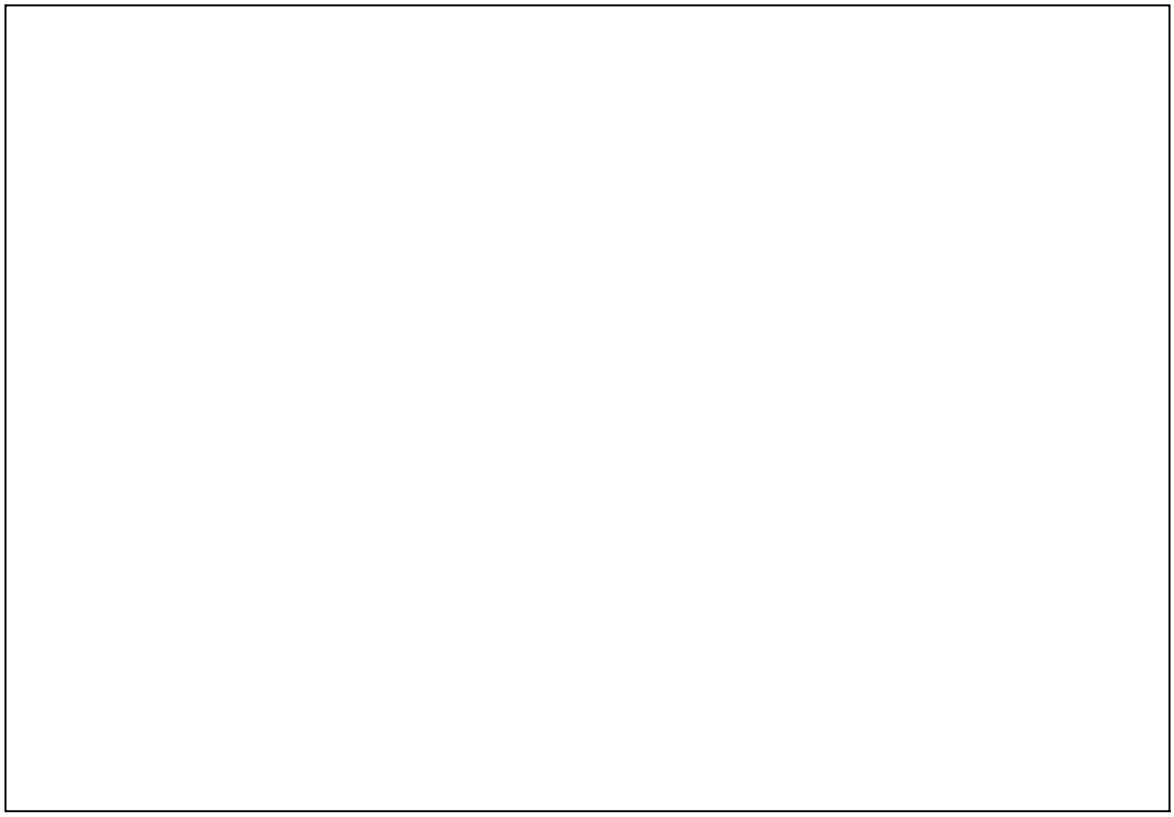
```
# echo eth0.7      > /etc/vservers/vs_7/interfaces/0/dev
# echo 10.0.7.0.2  > /etc/vservers/vs_7/interfaces/0/ip
# echo eth0.8      > /etc/vservers/vs_8/interfaces/0/dev
# echo 10.0.8.0.2  > /etc/vservers/vs_8/interfaces/0/ip
```

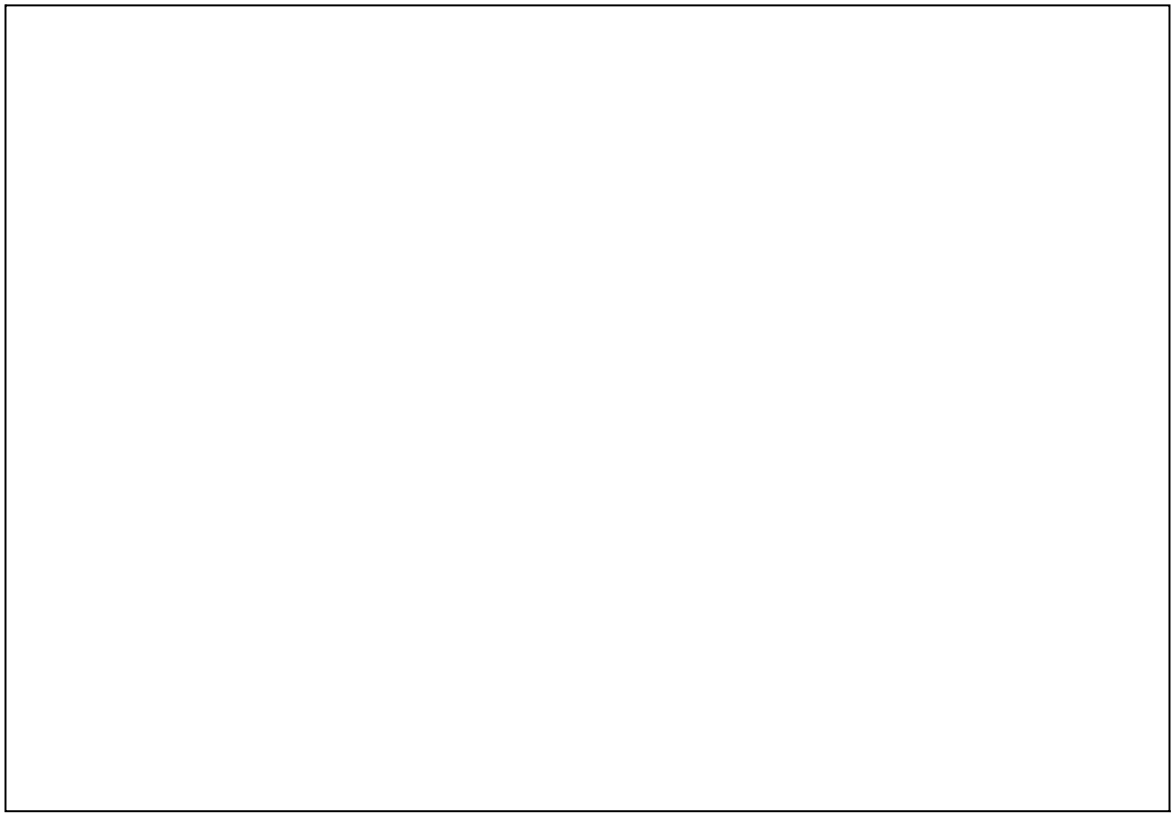


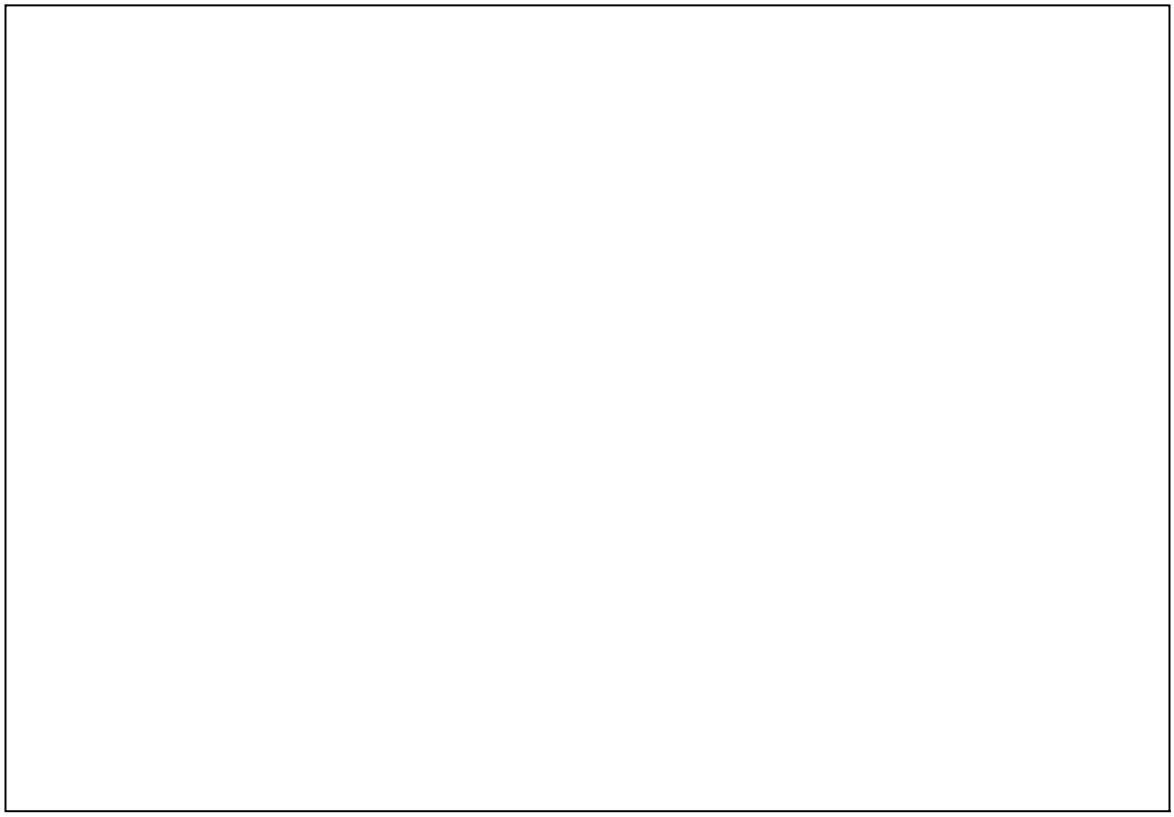


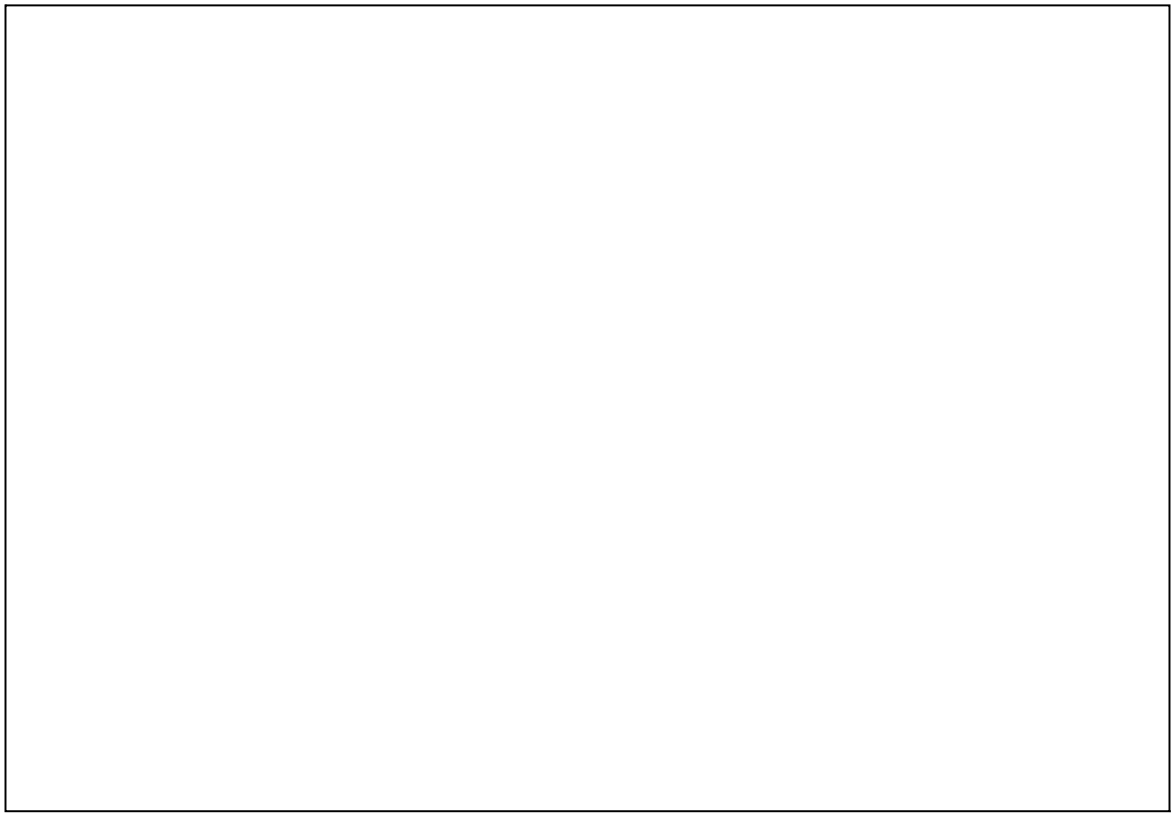


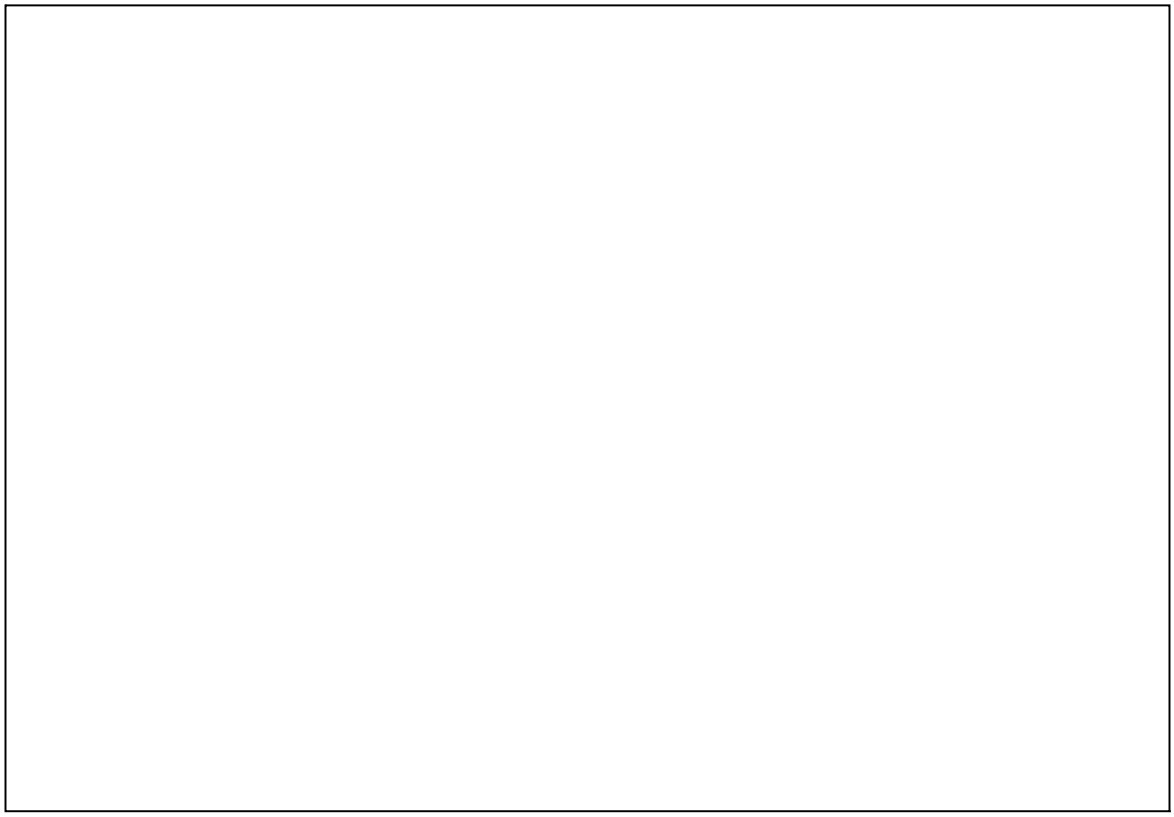


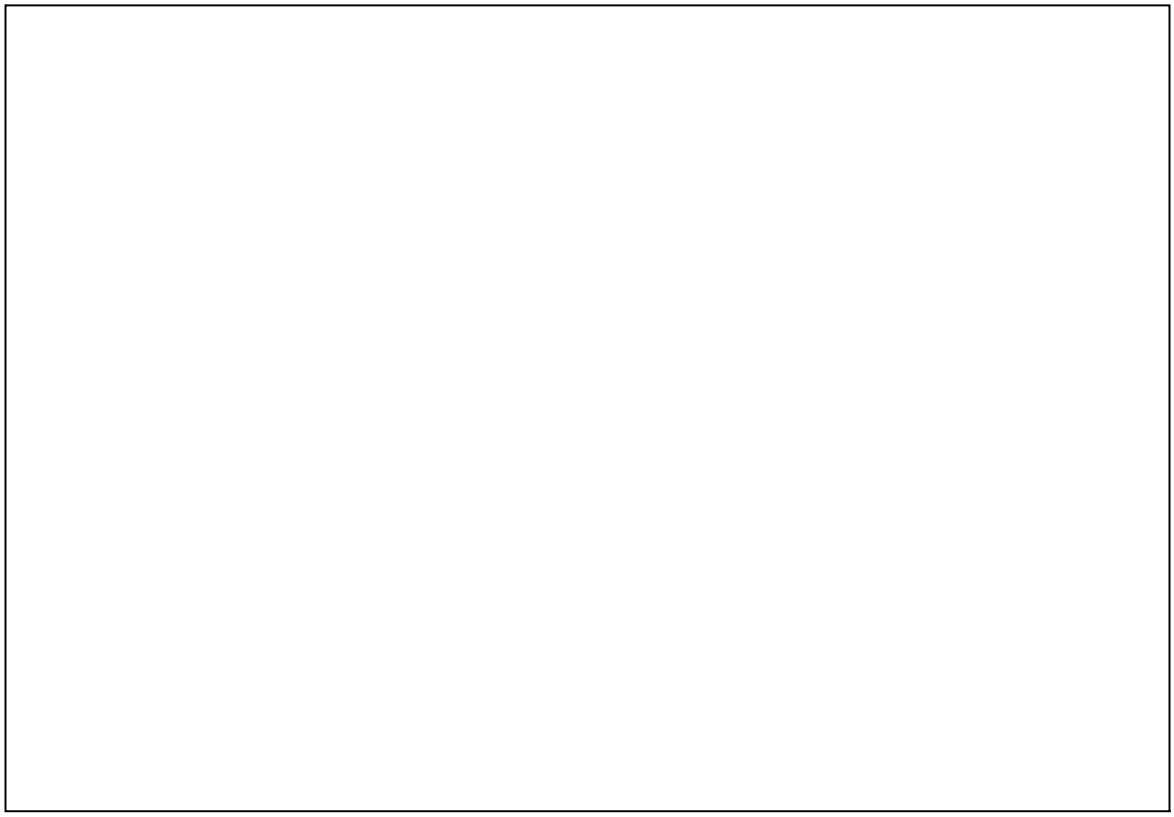


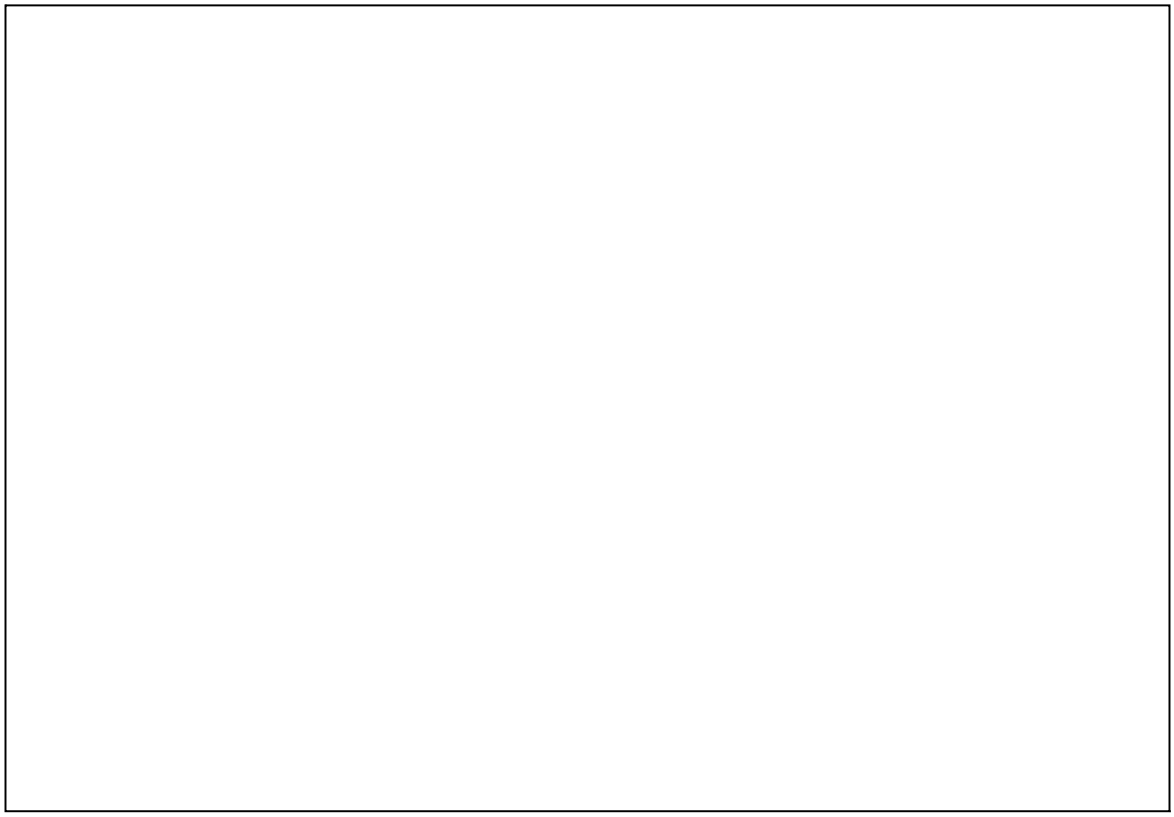


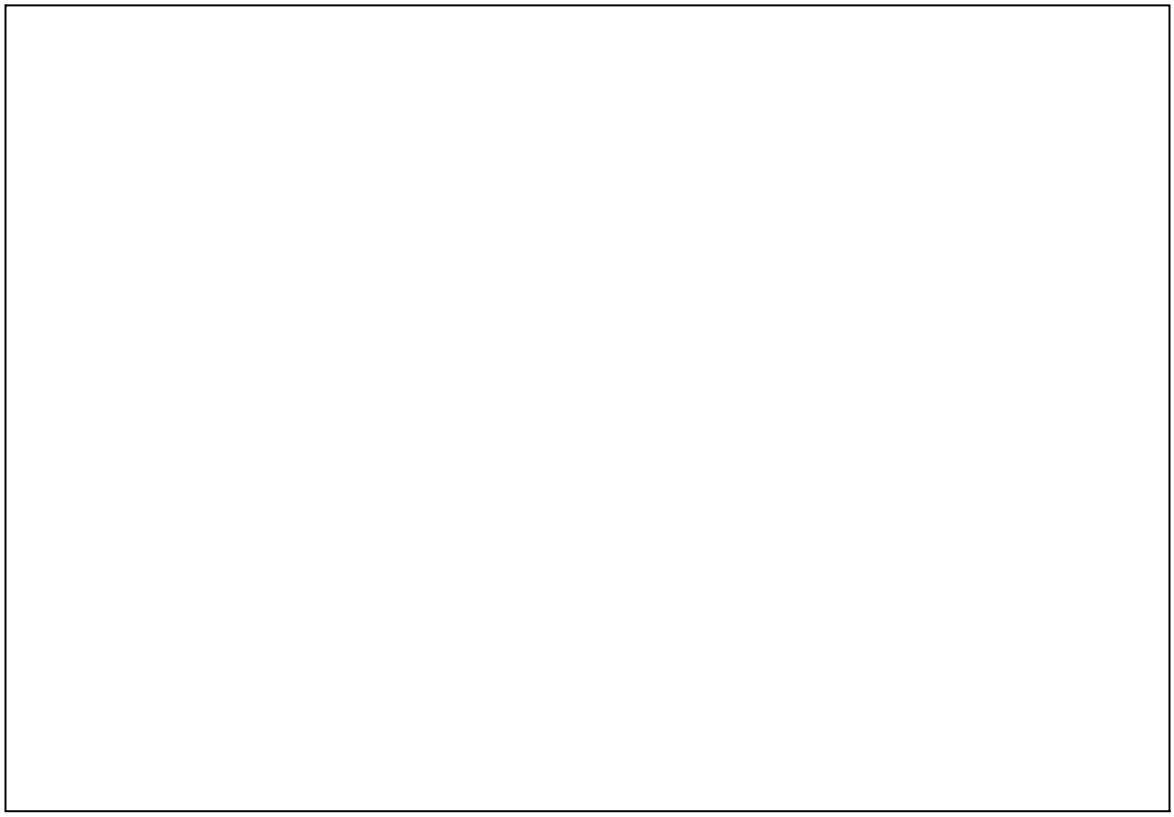


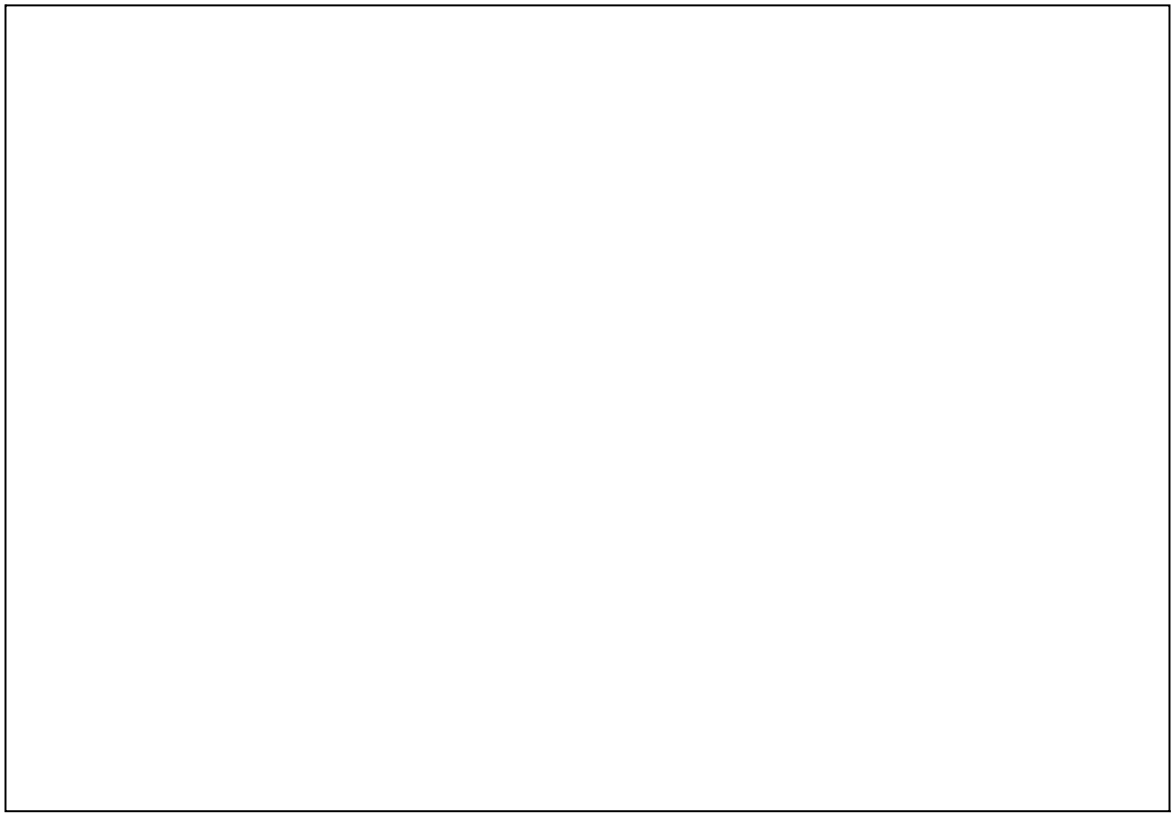


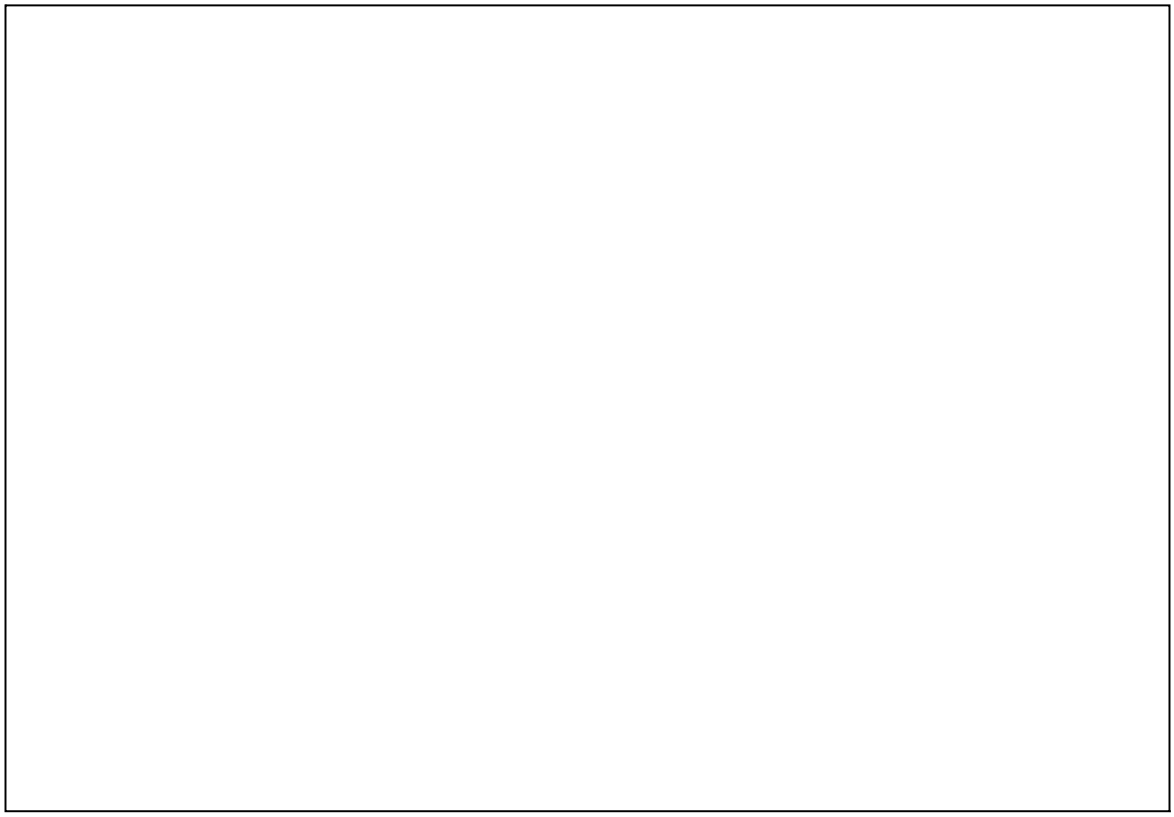


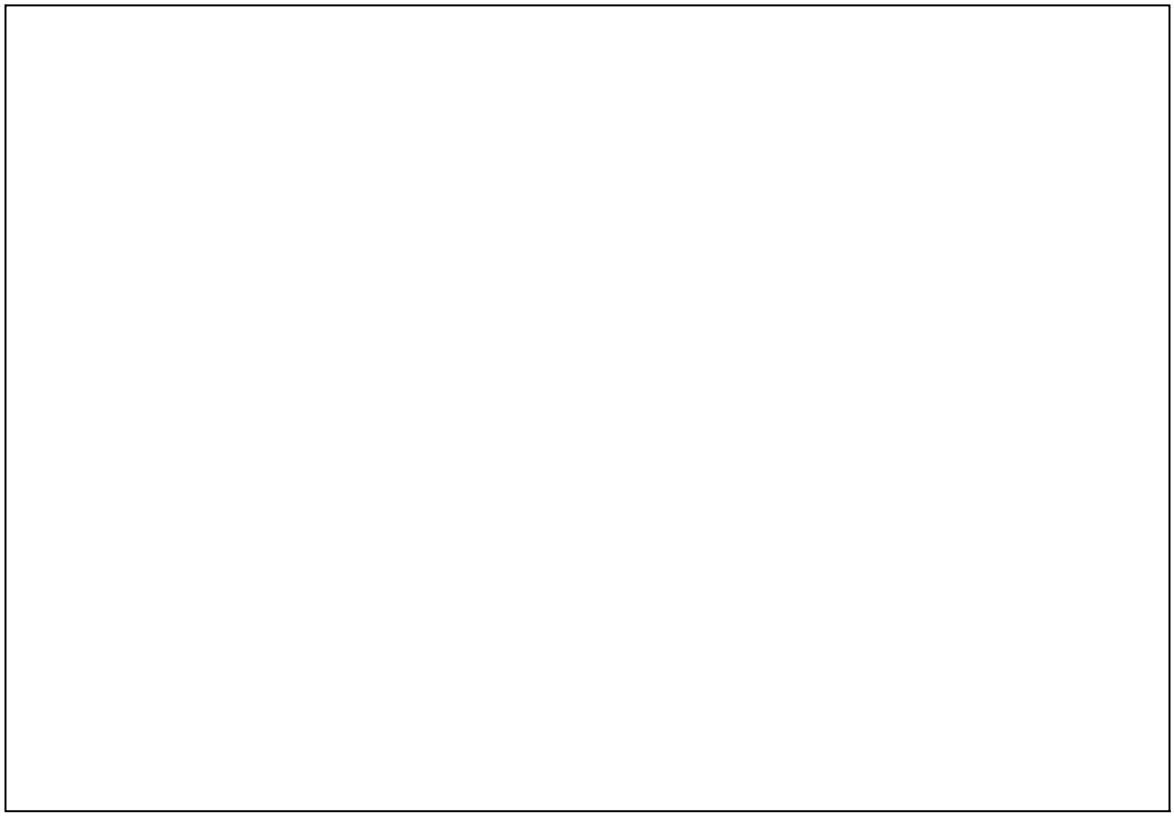


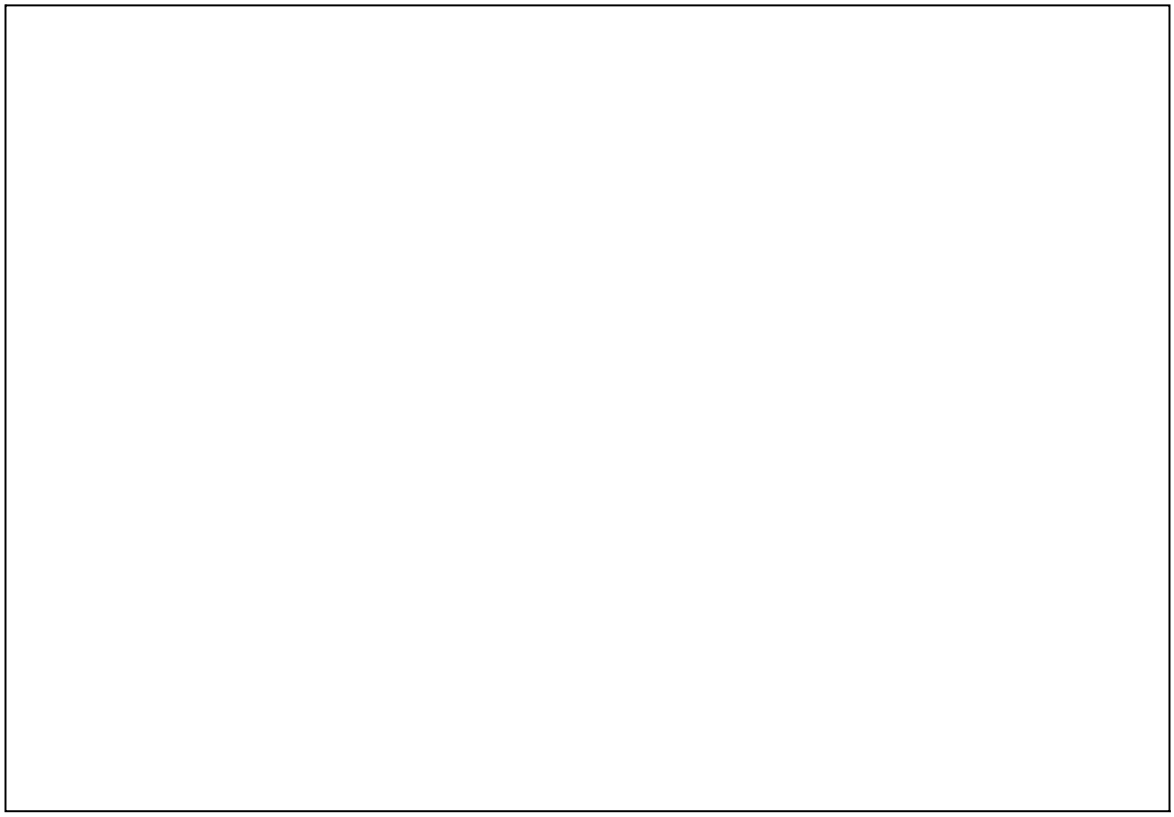


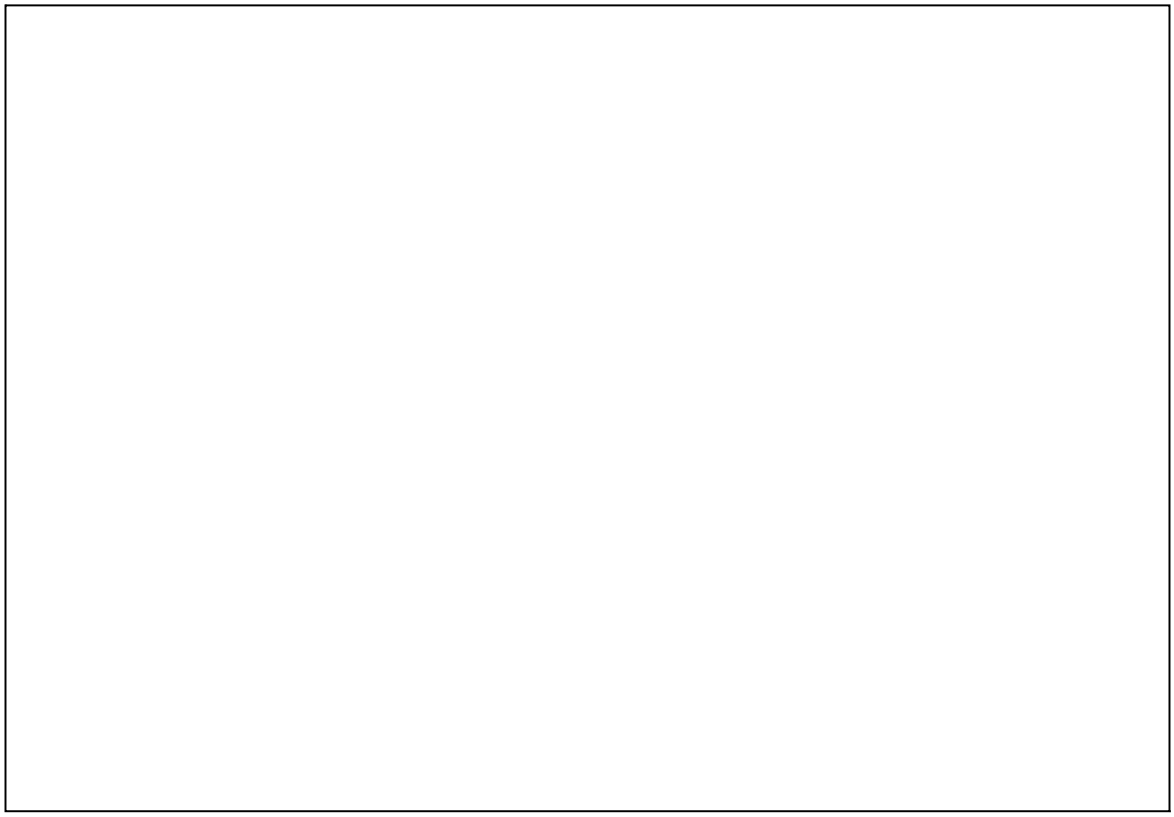


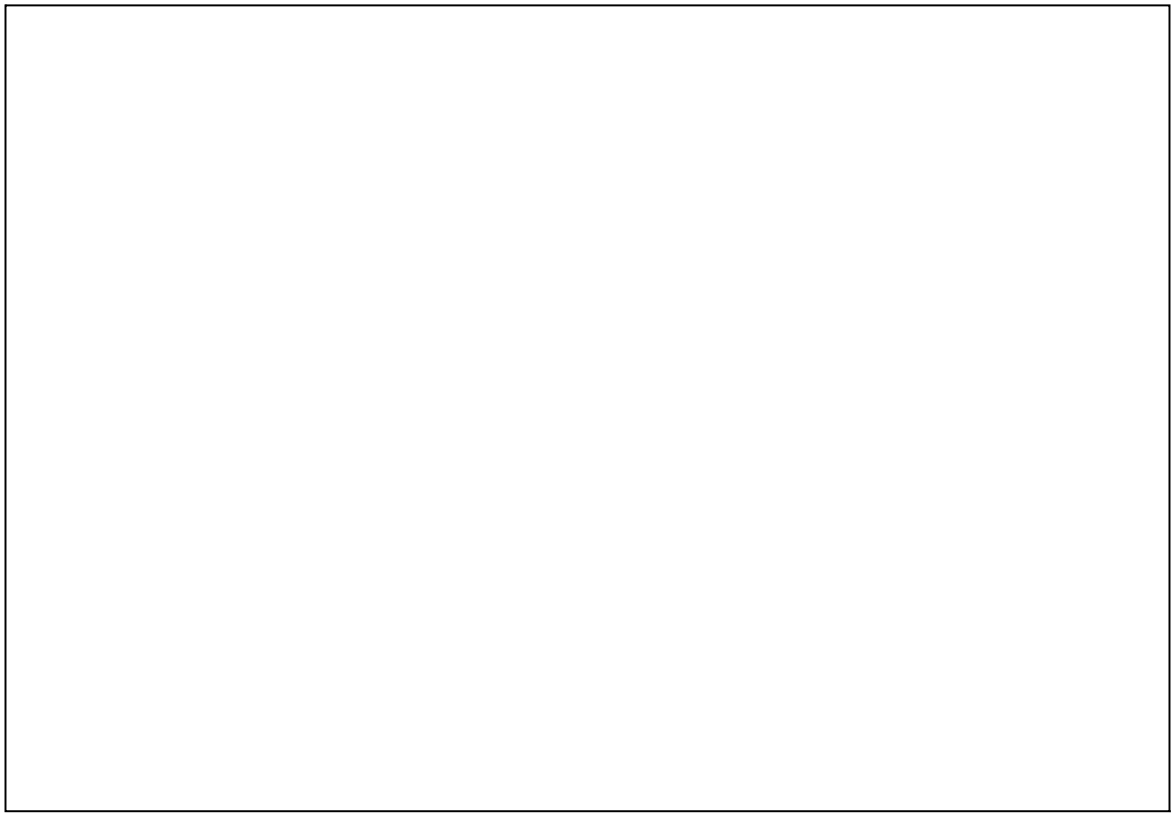


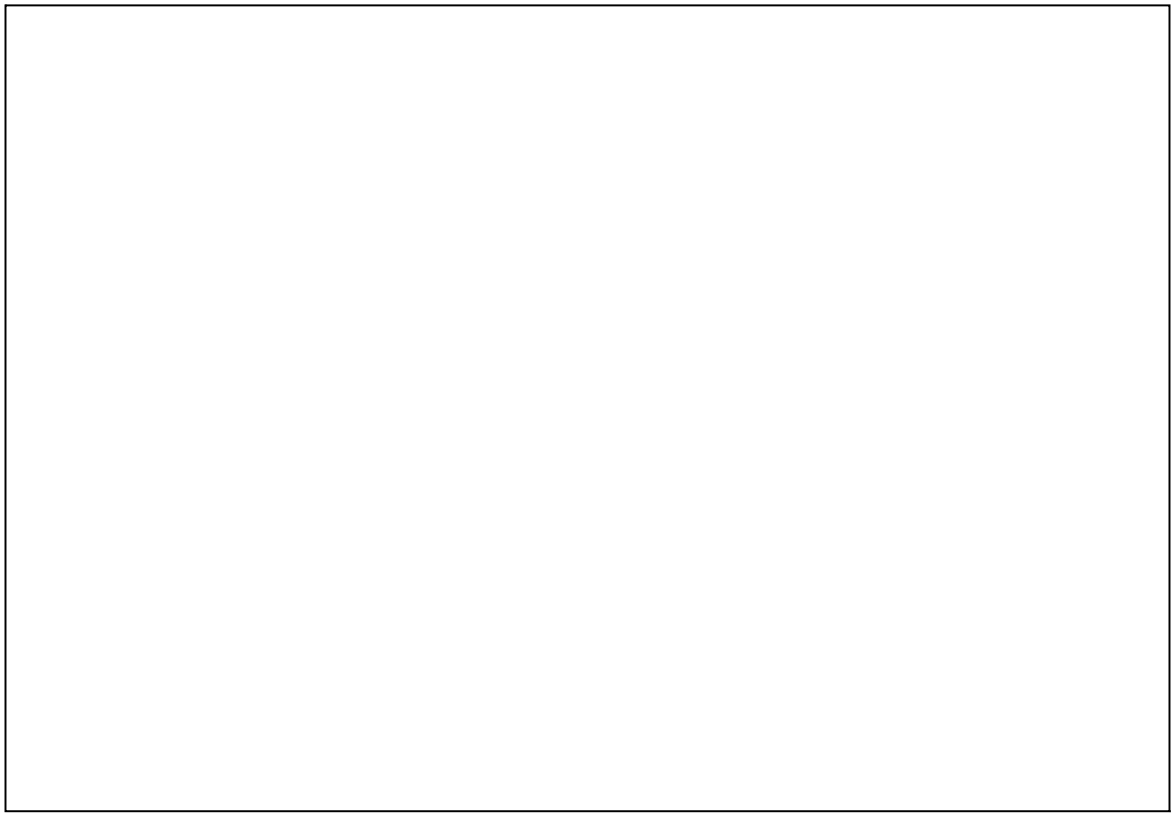


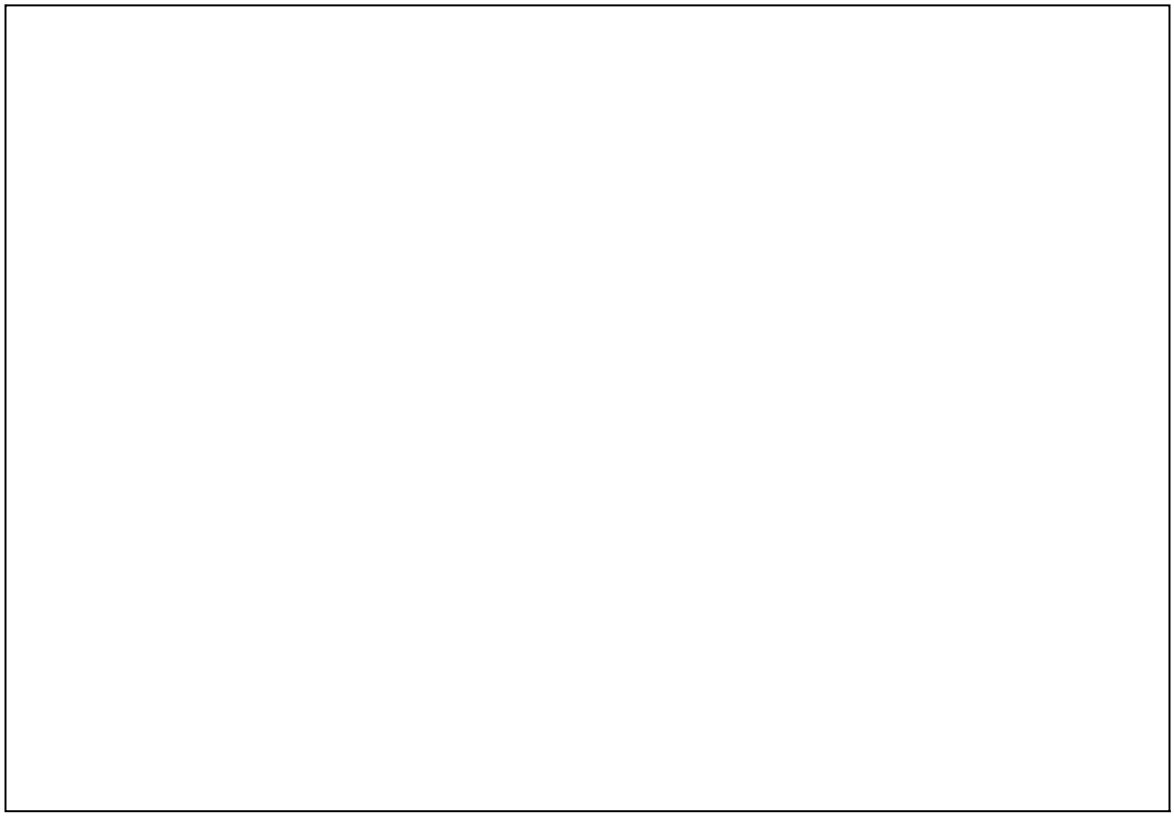


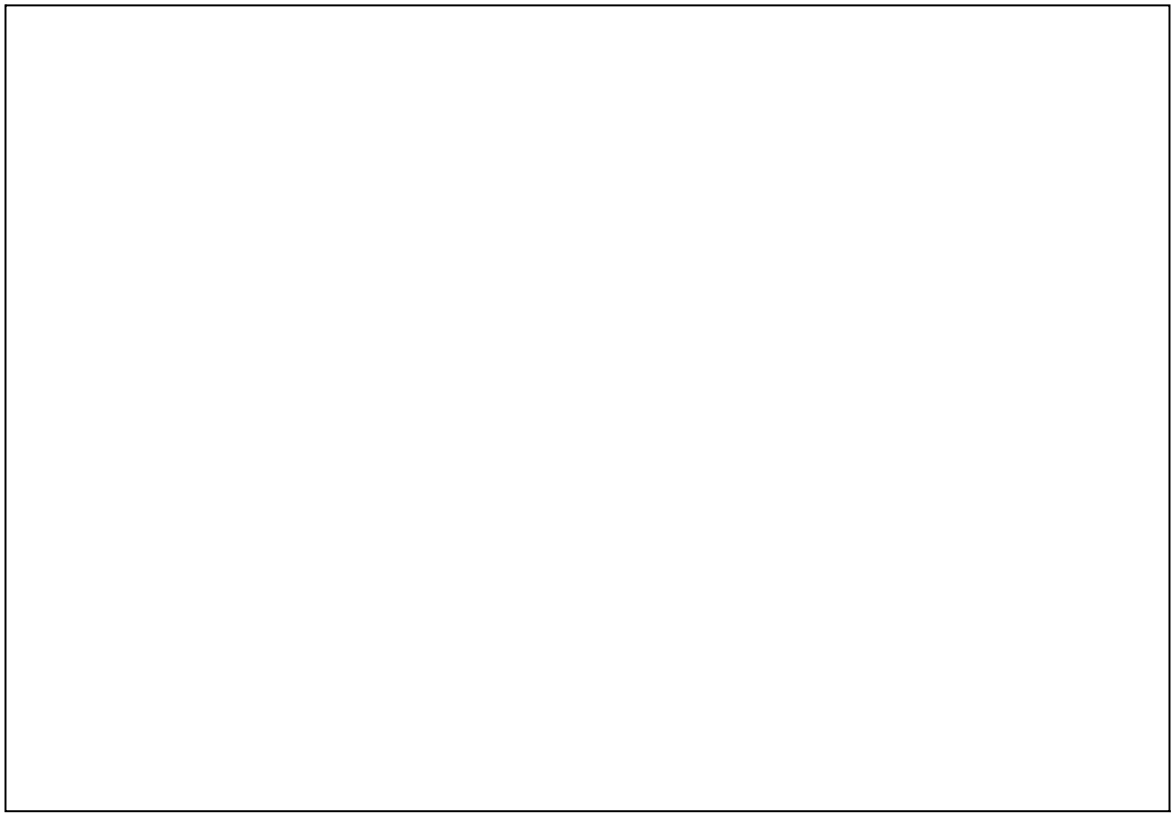


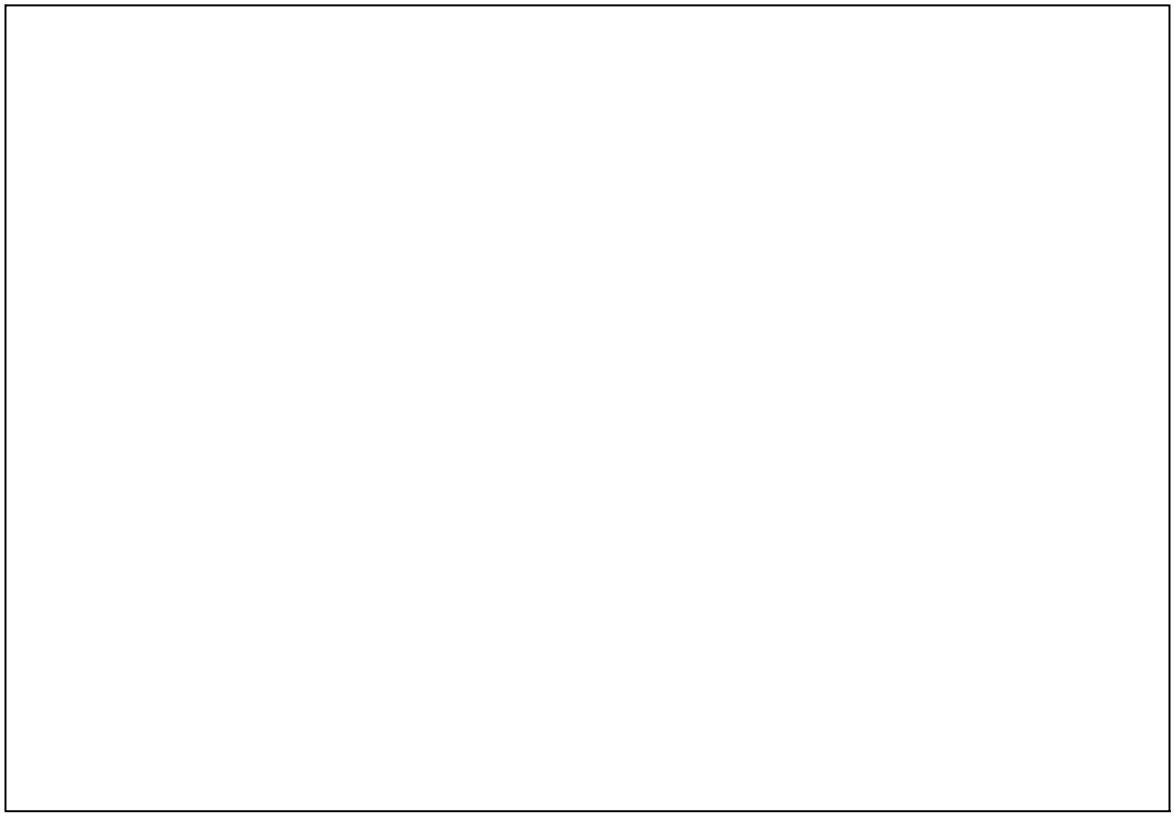


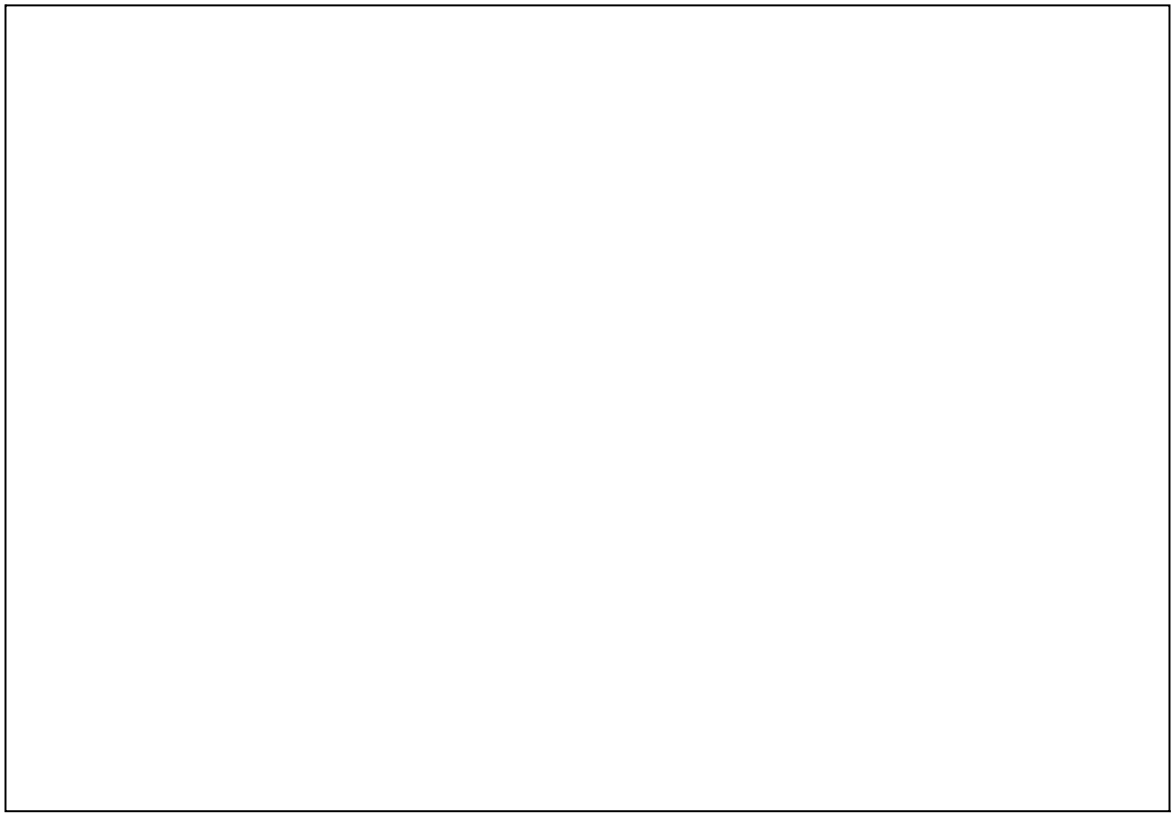


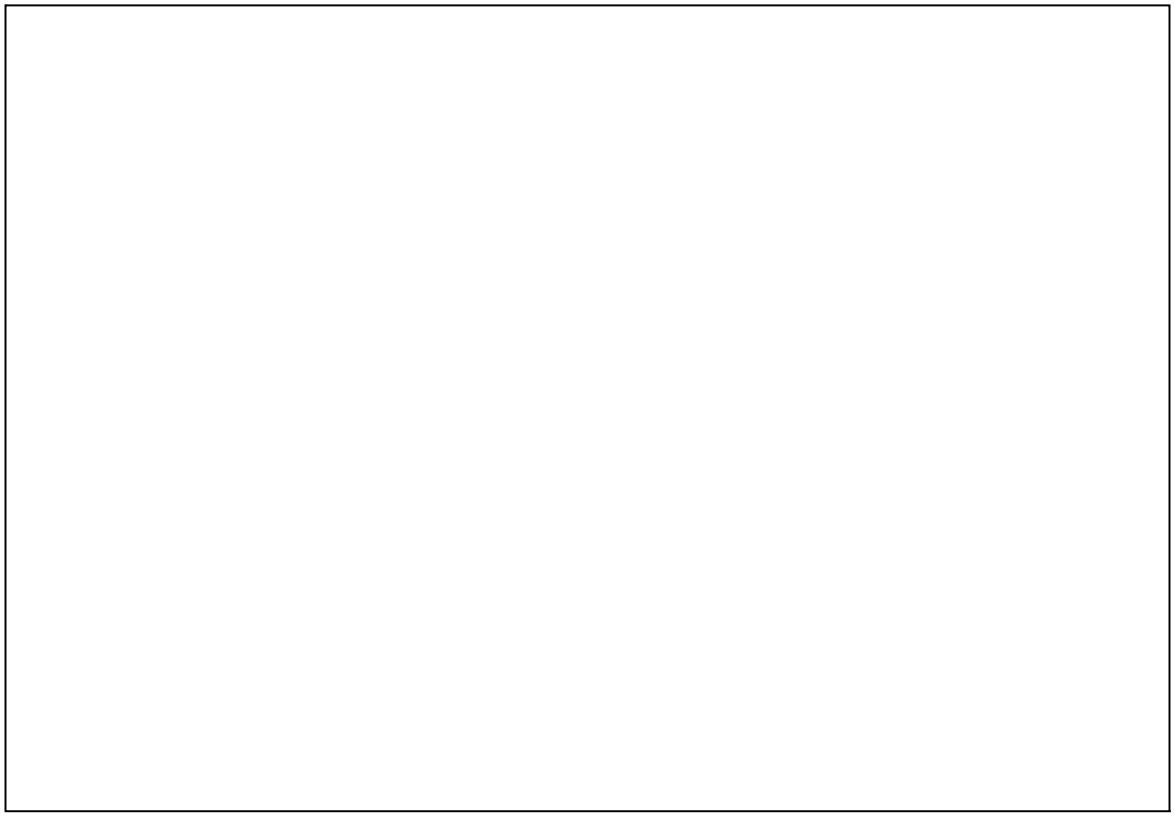


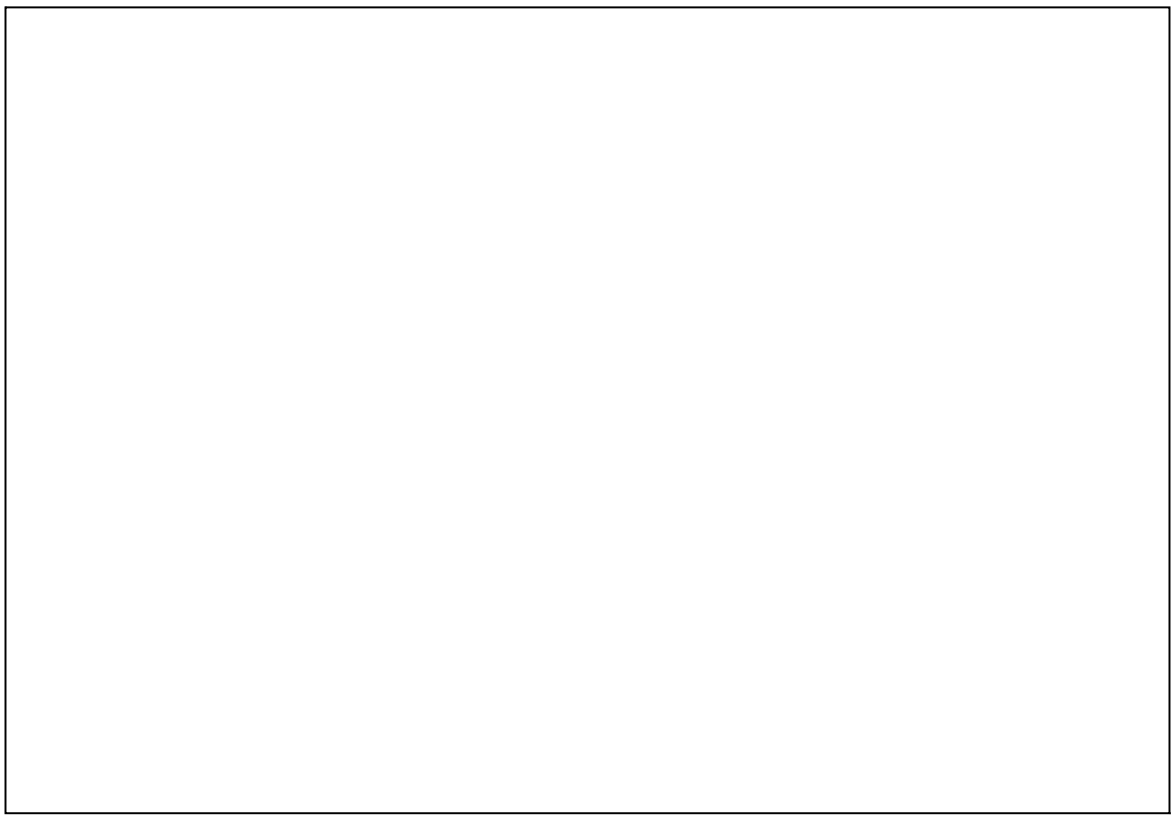


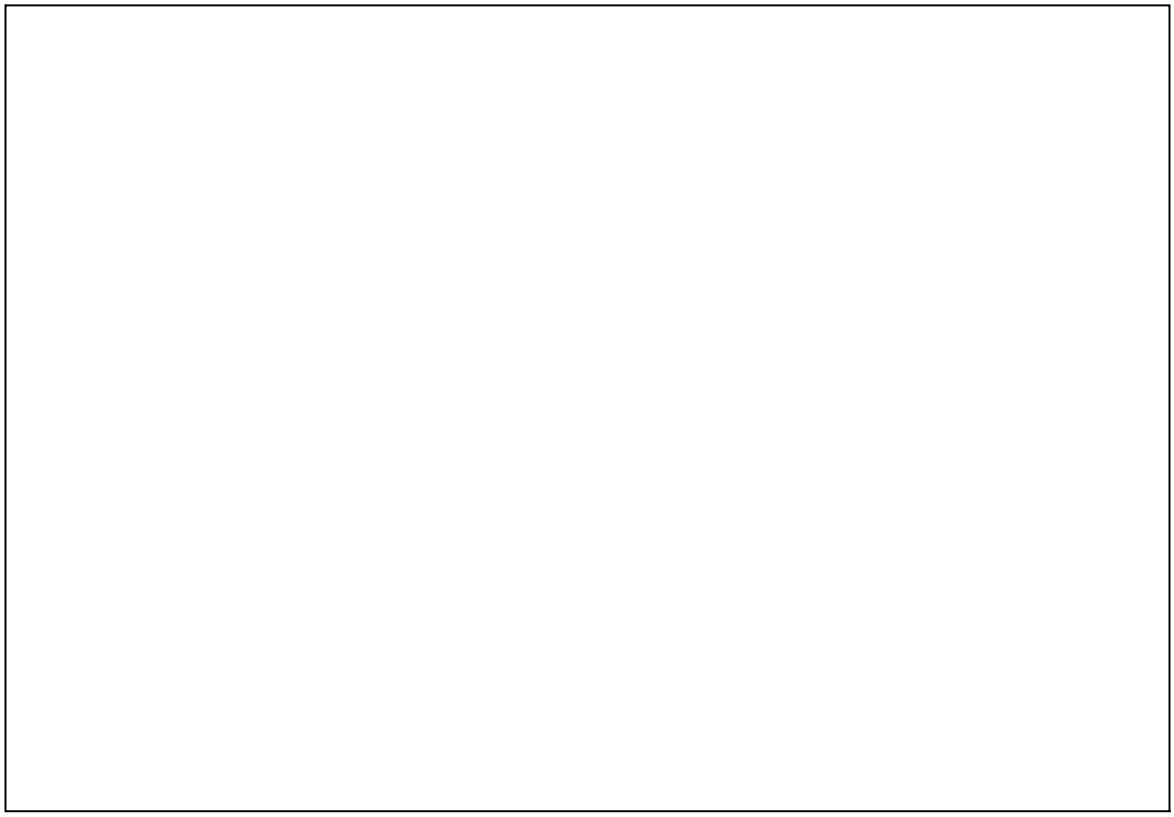


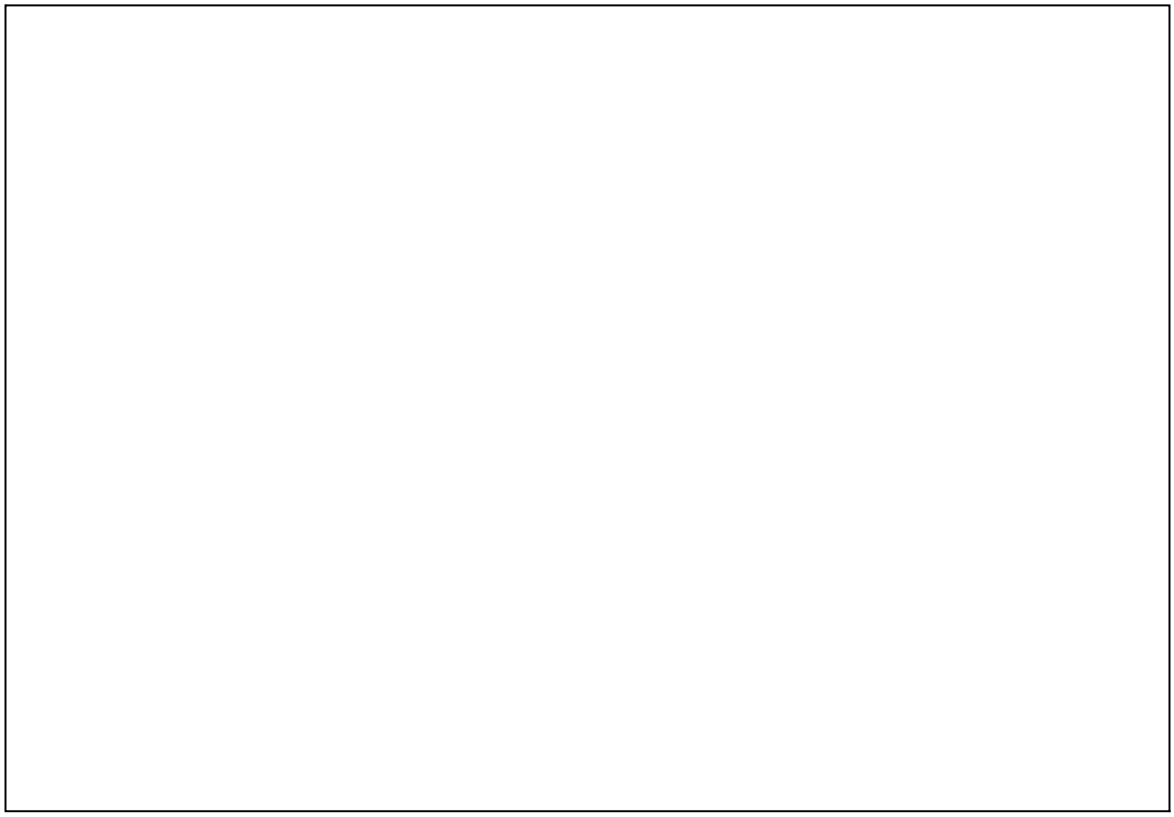


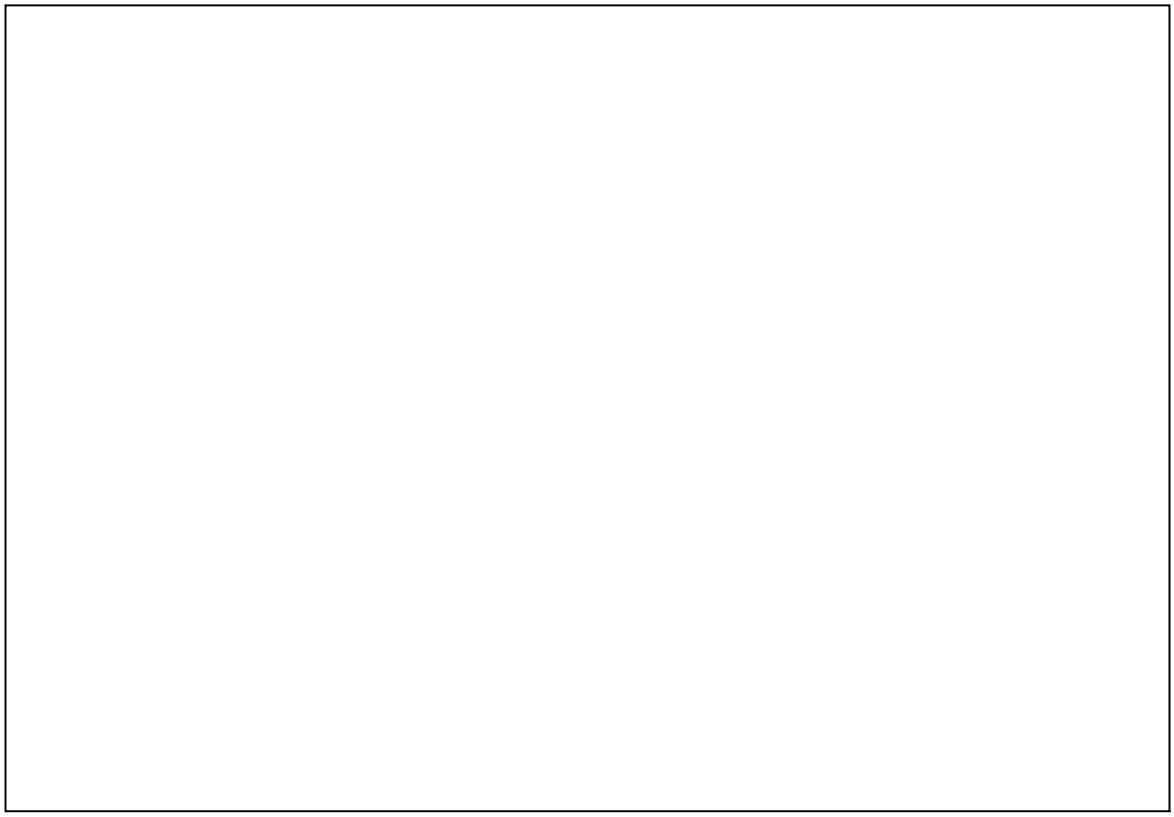


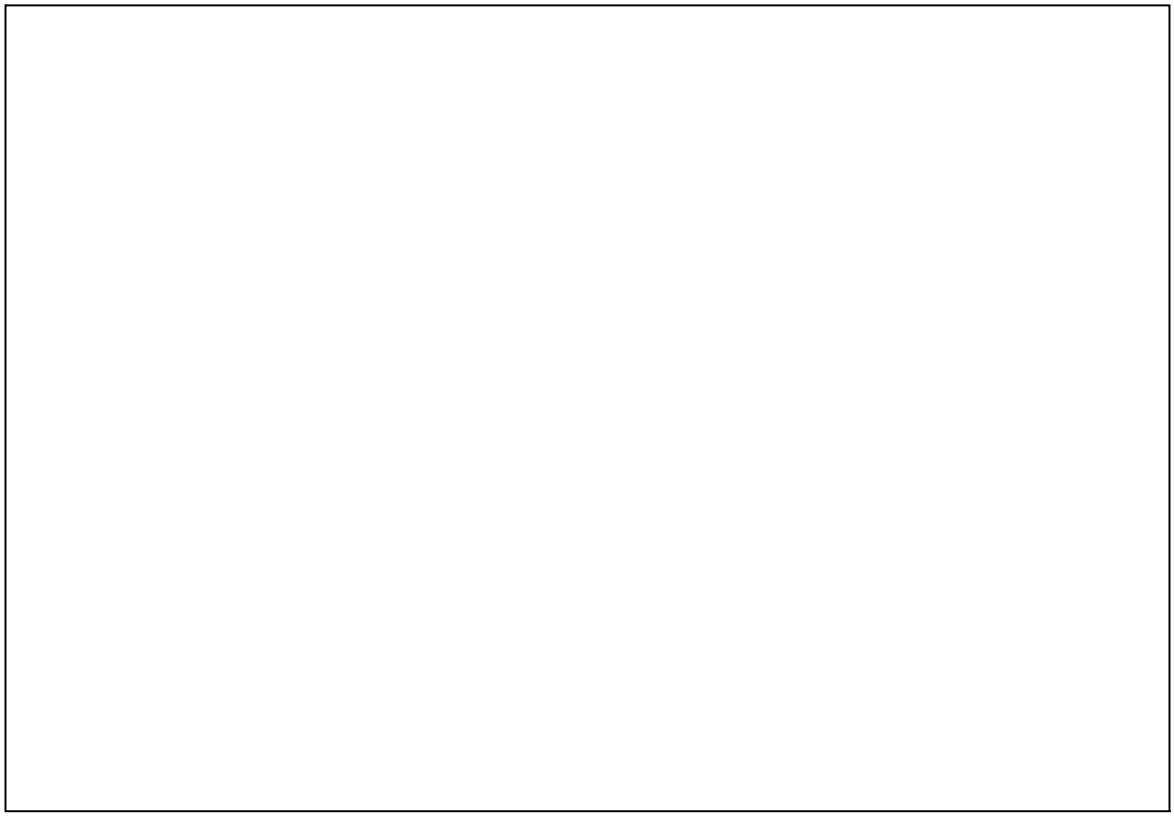


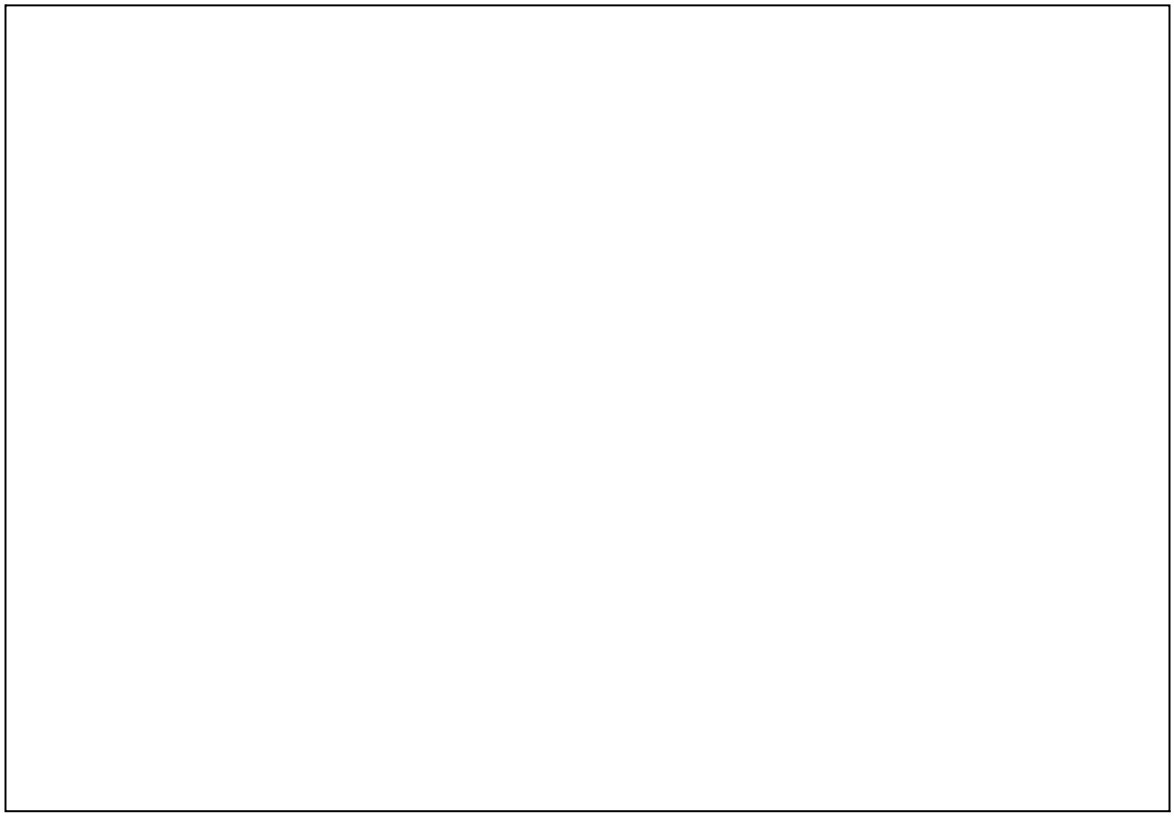


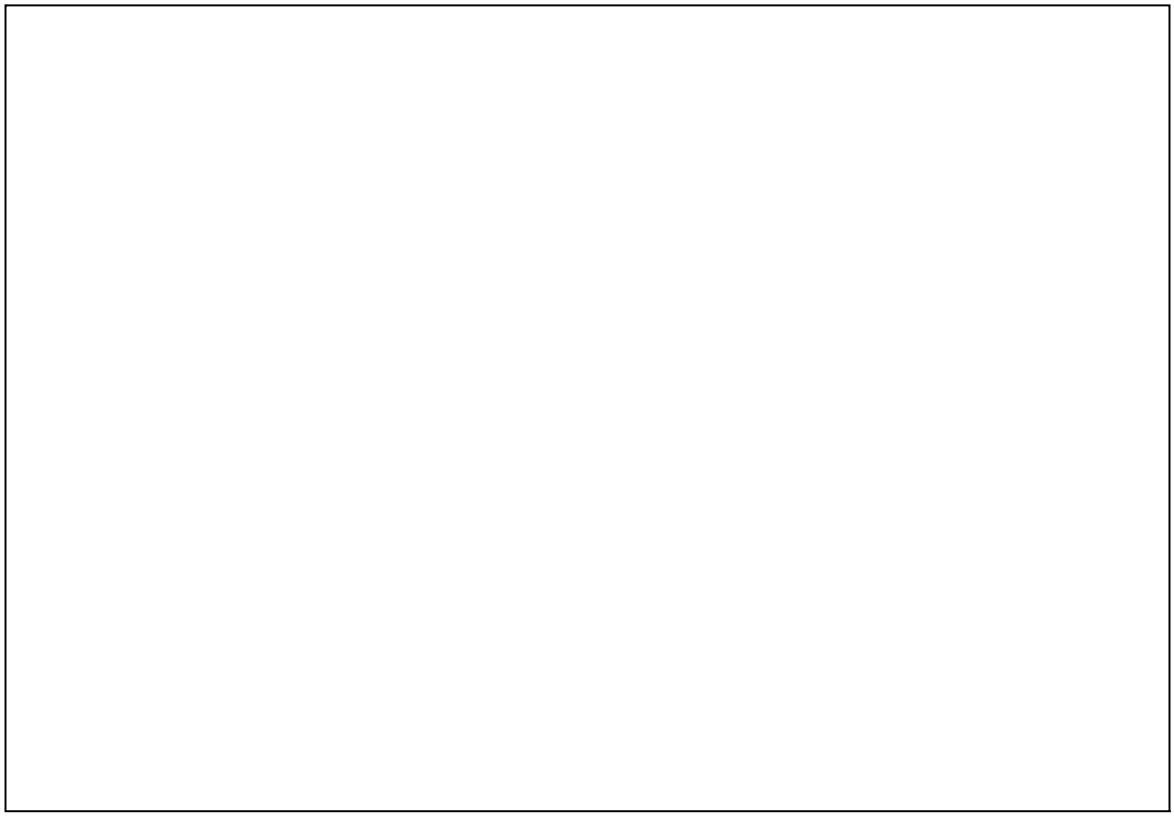


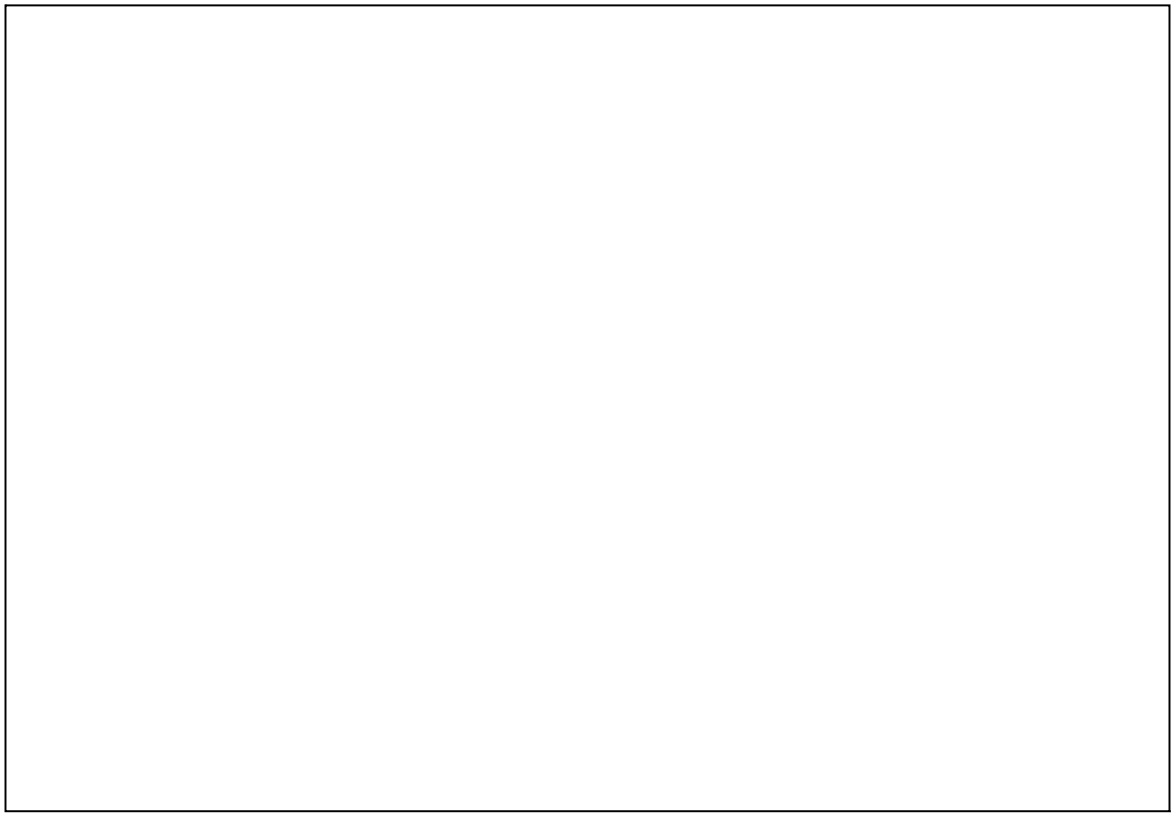


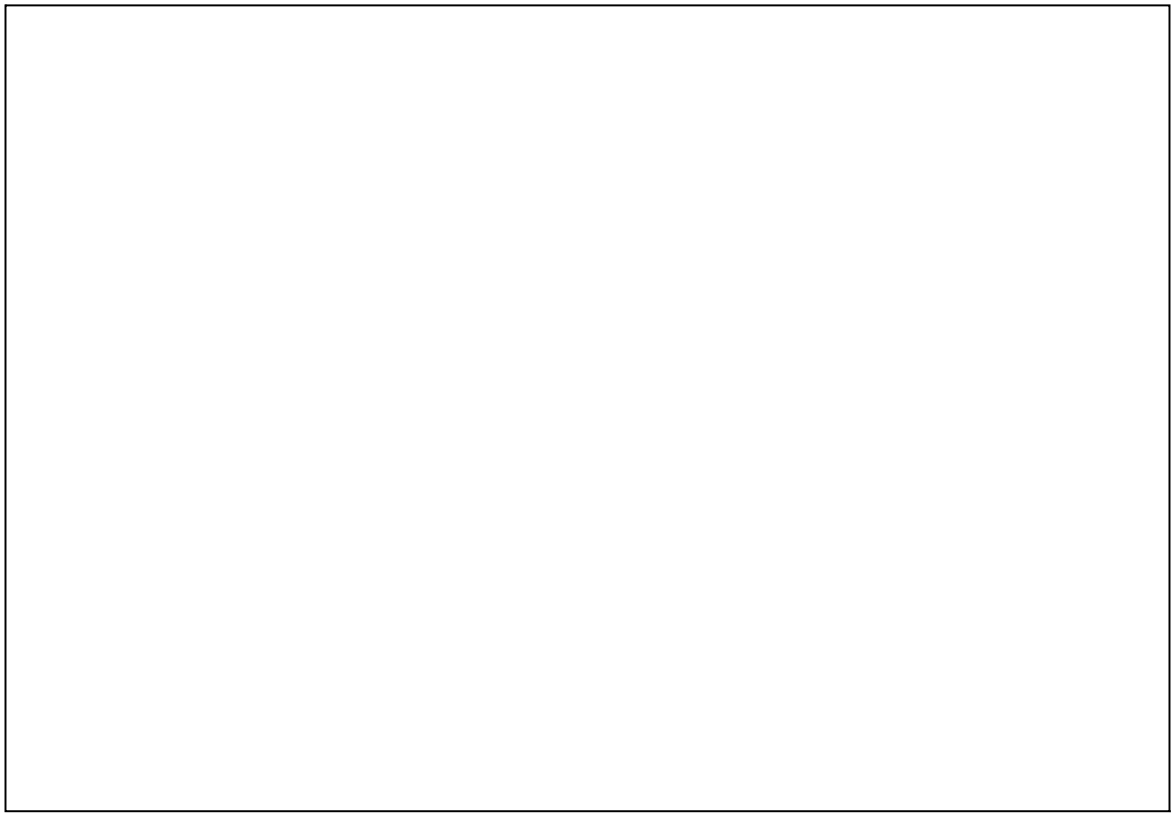


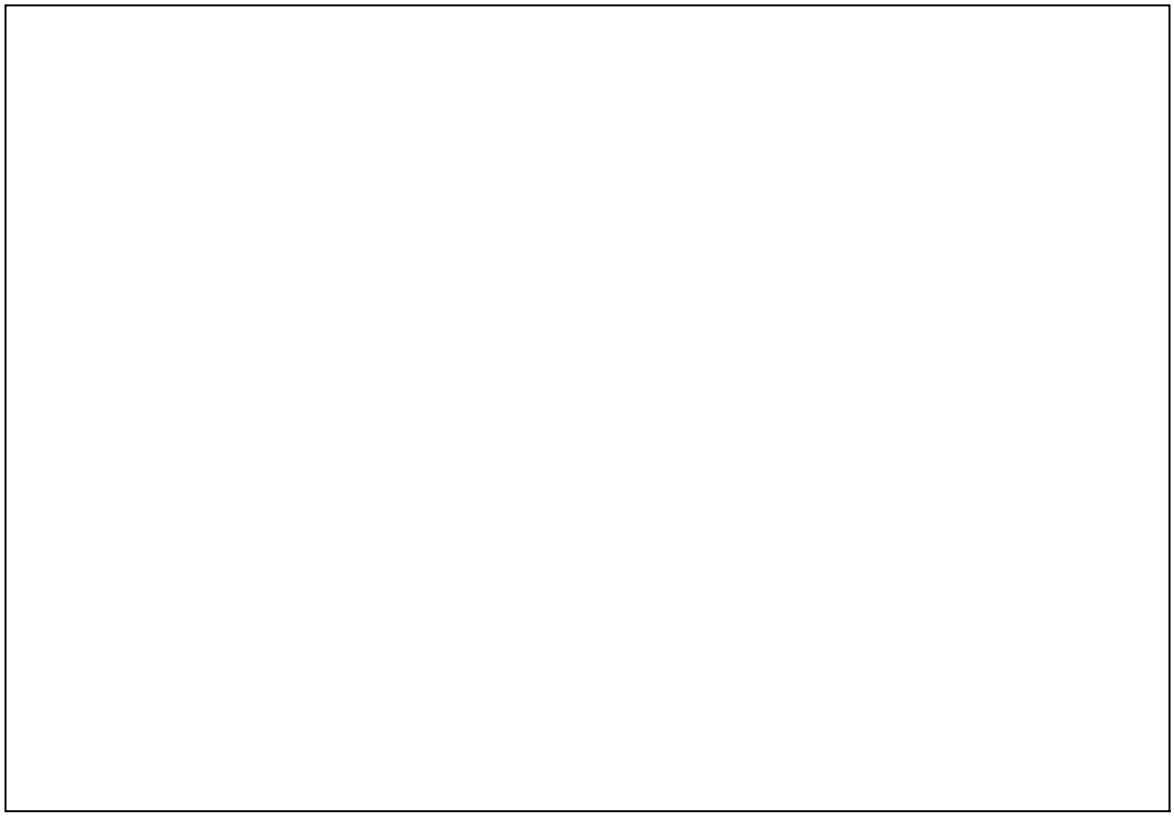


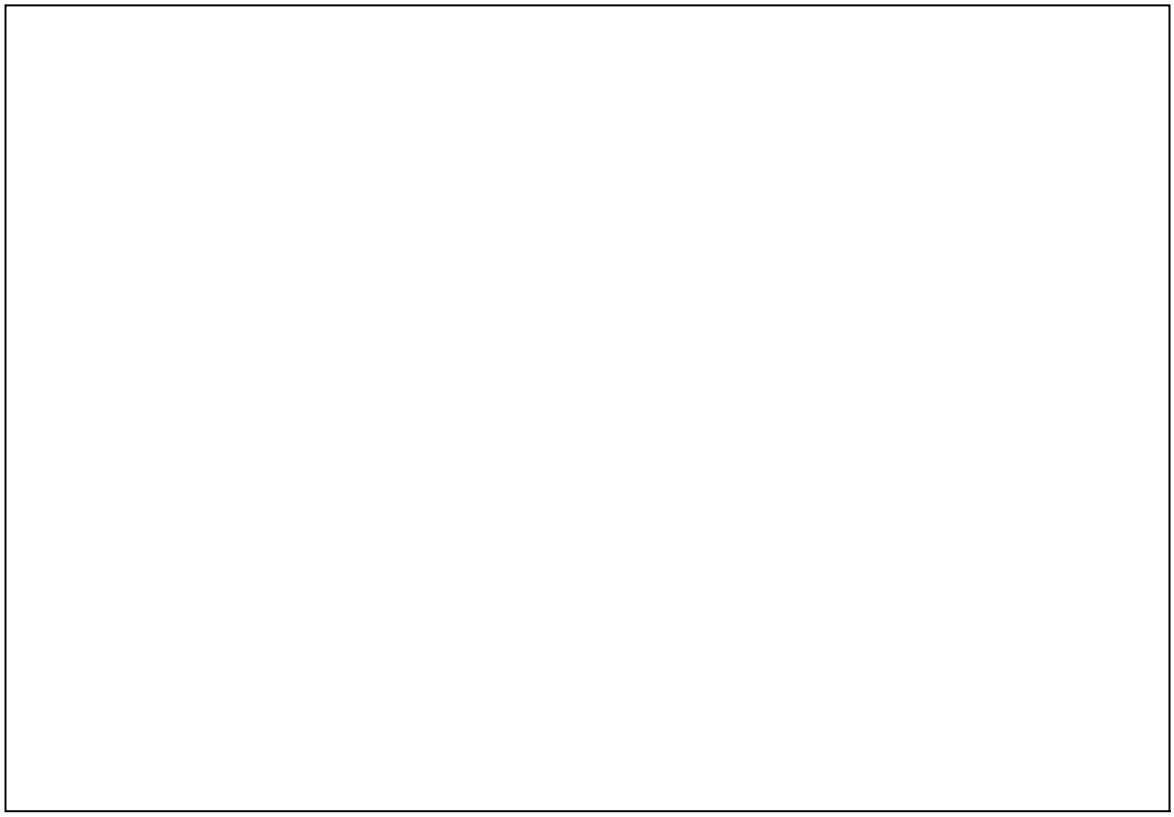


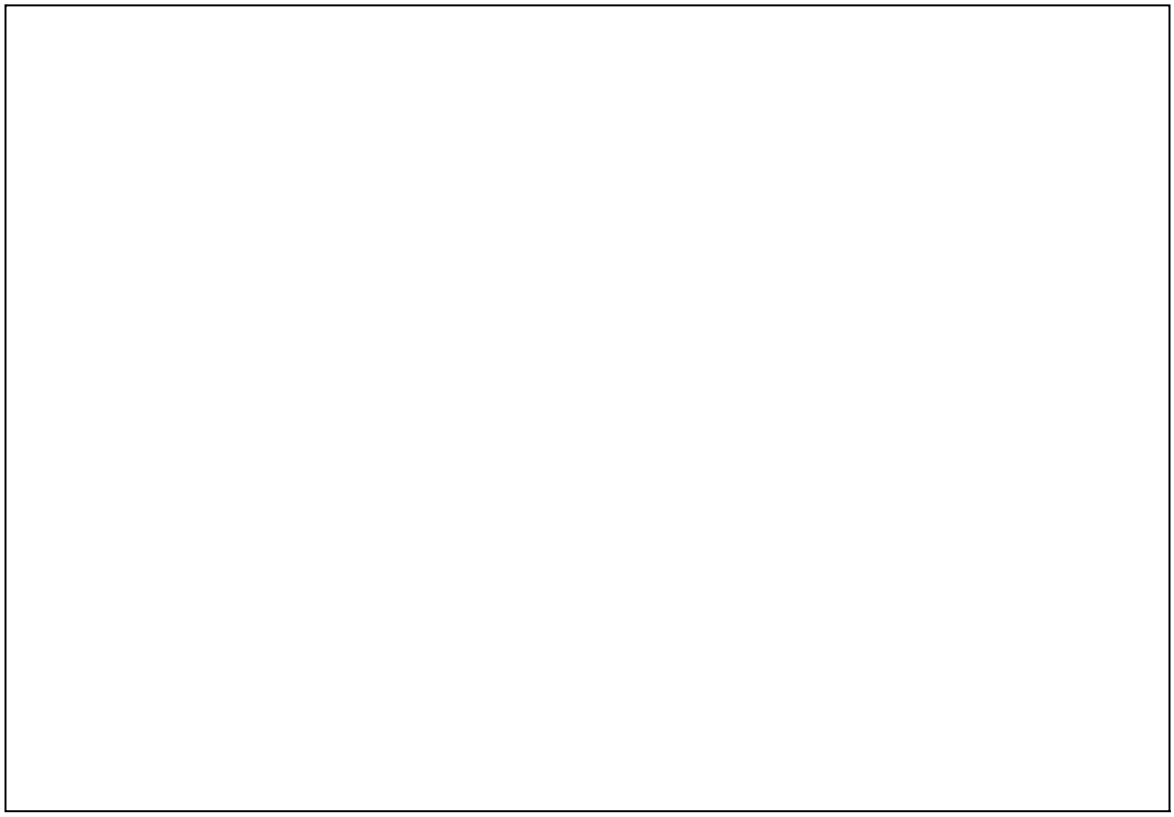


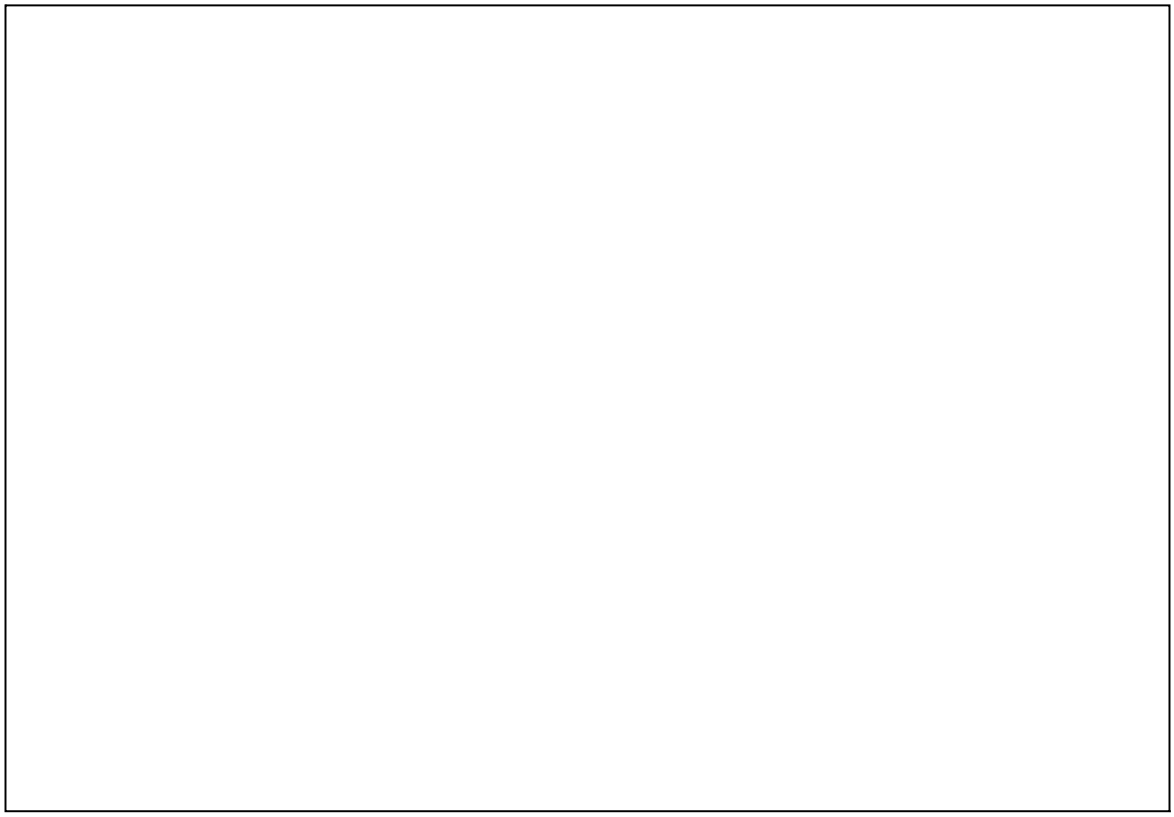


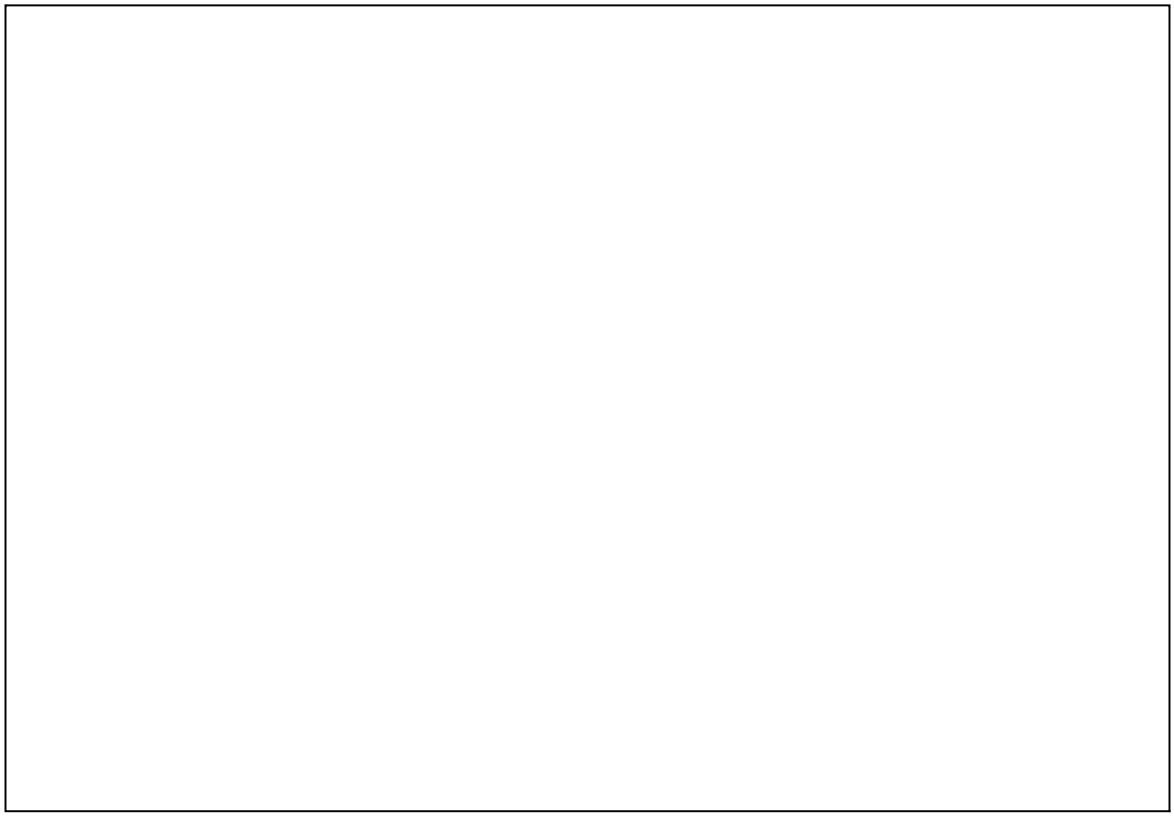


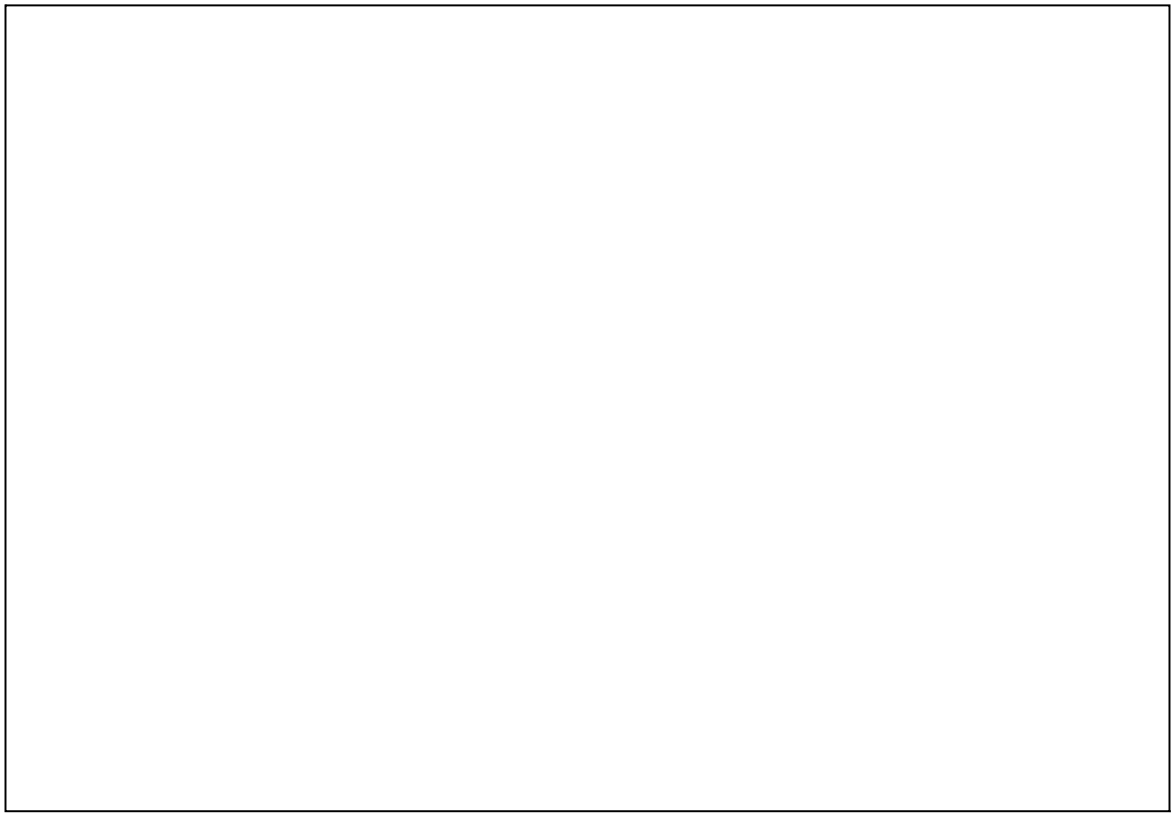


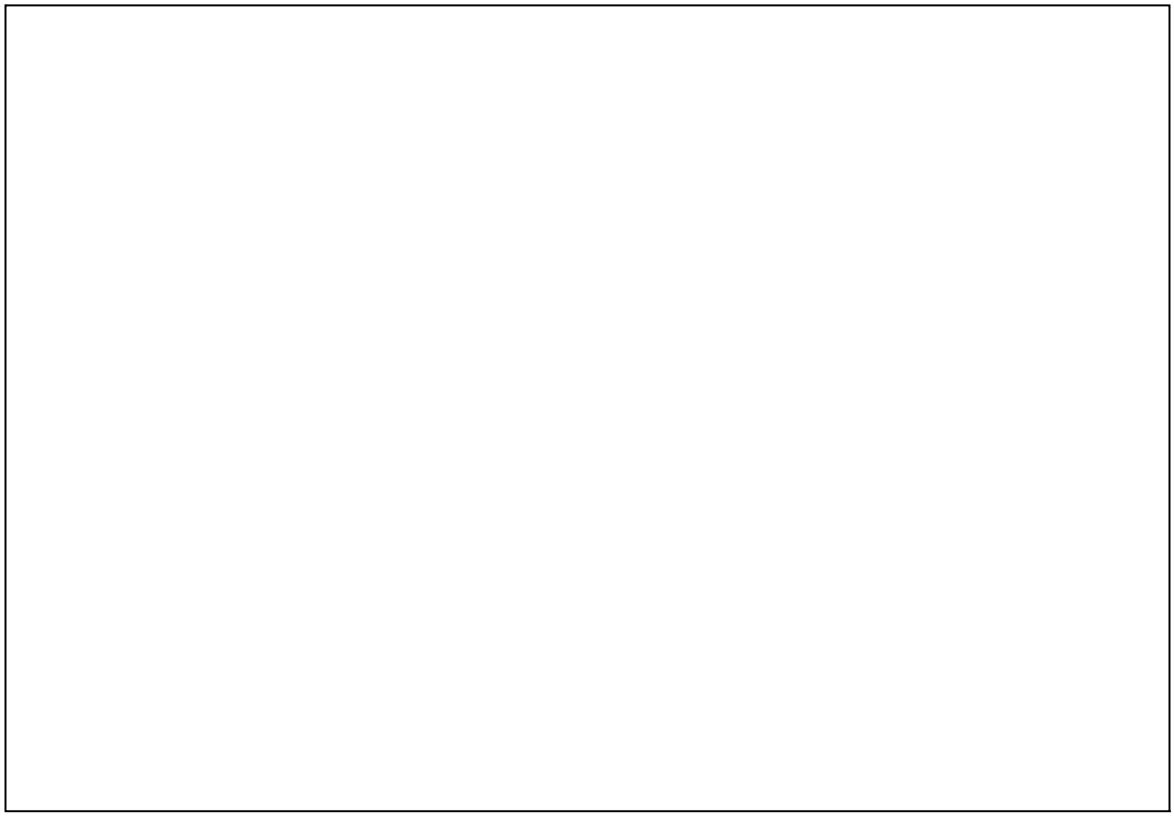


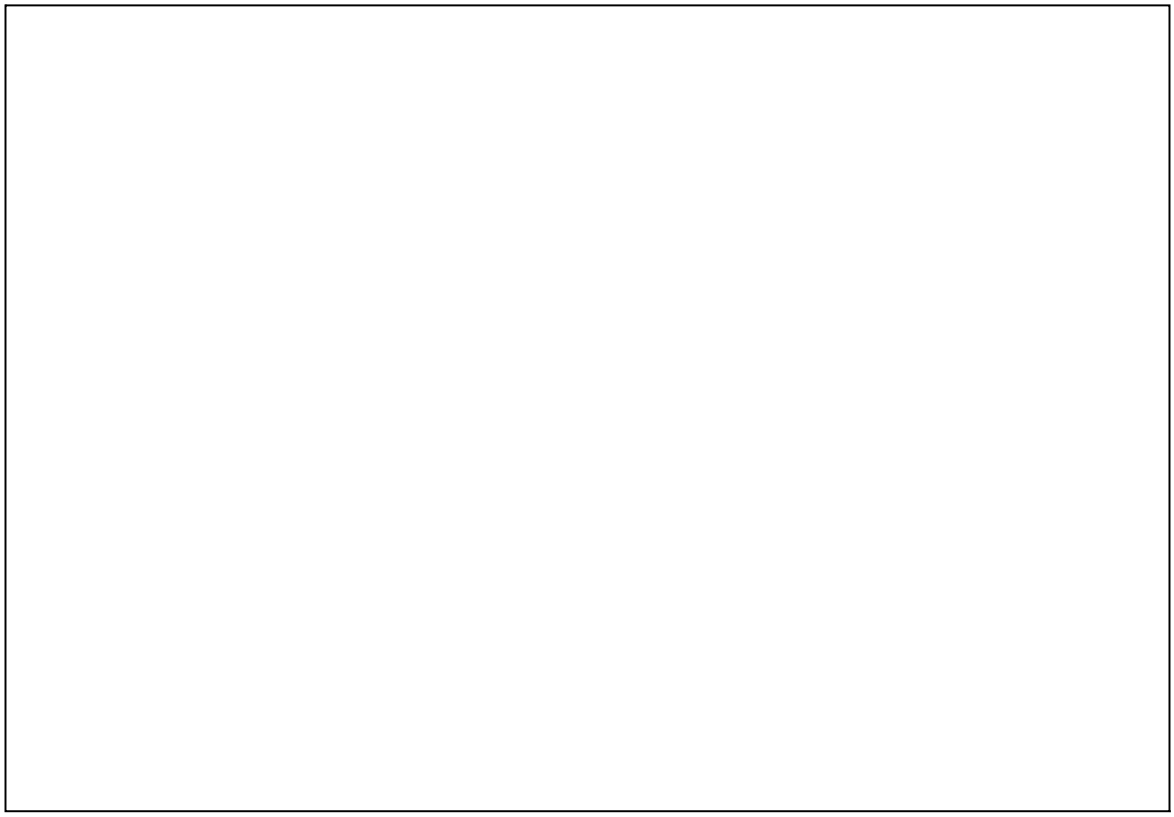


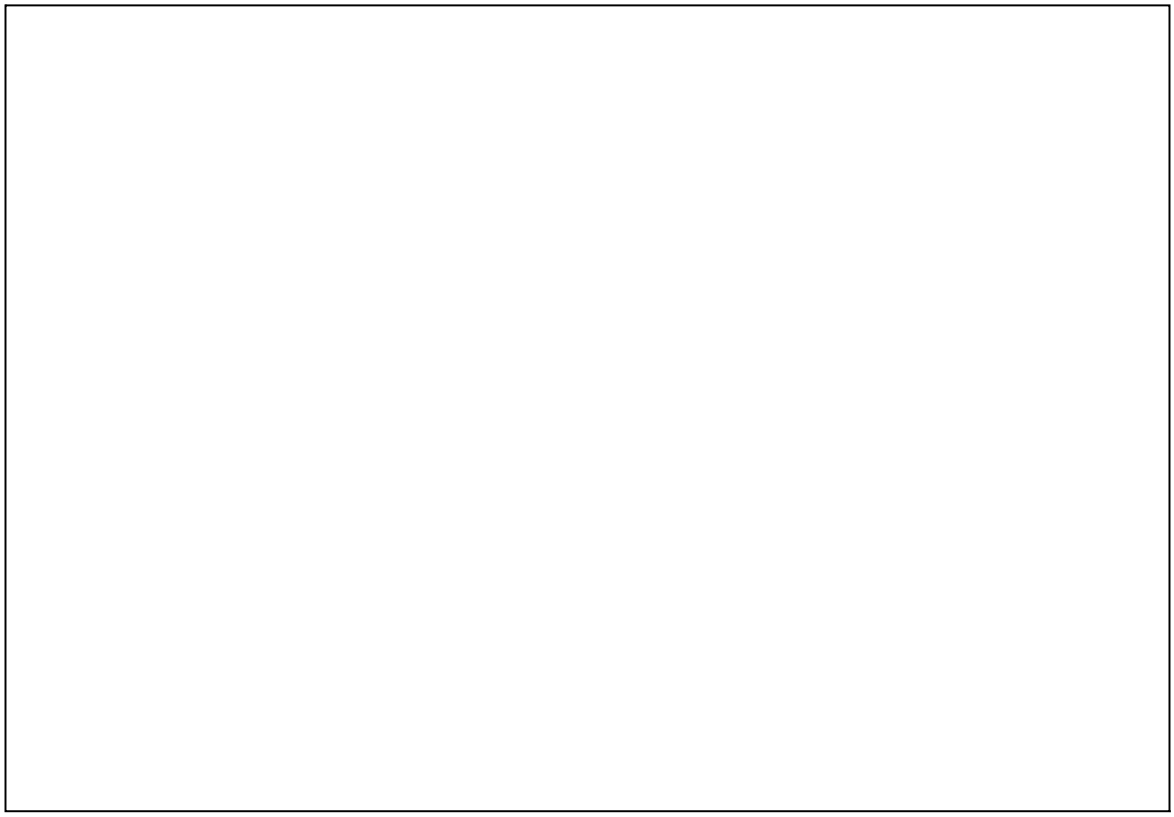


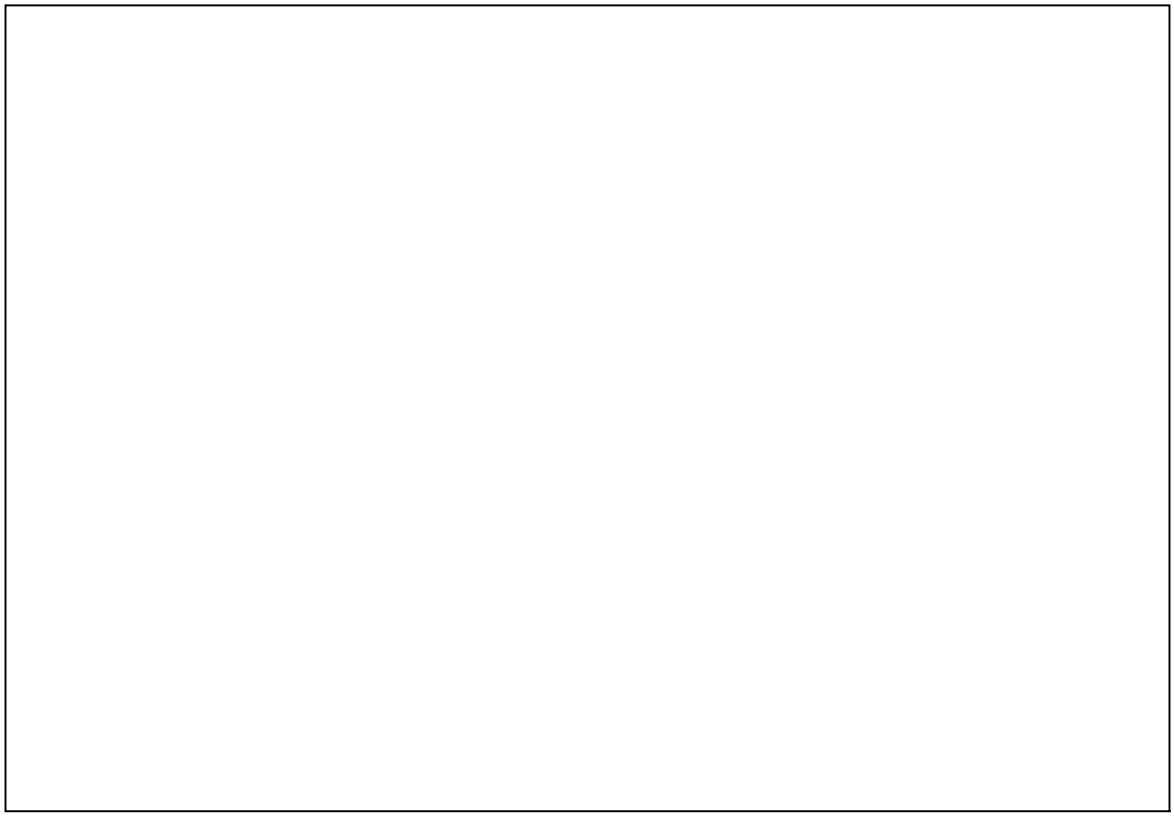


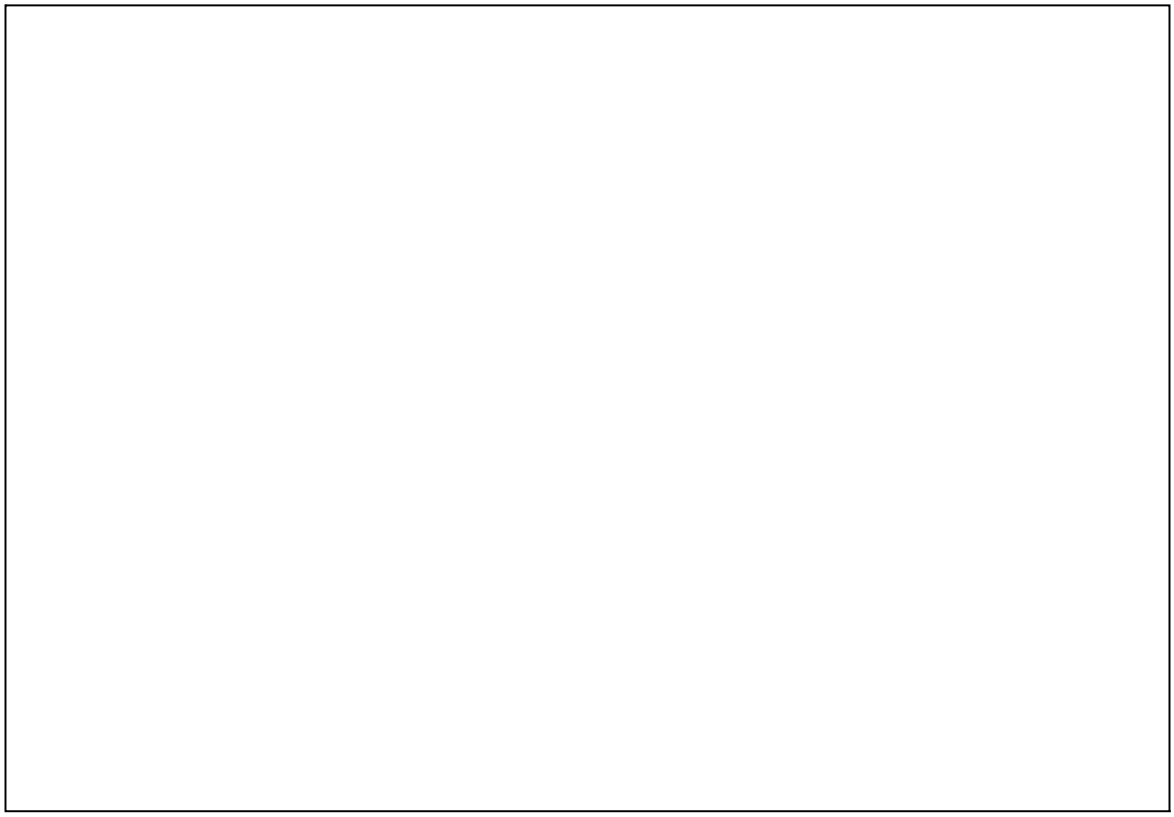


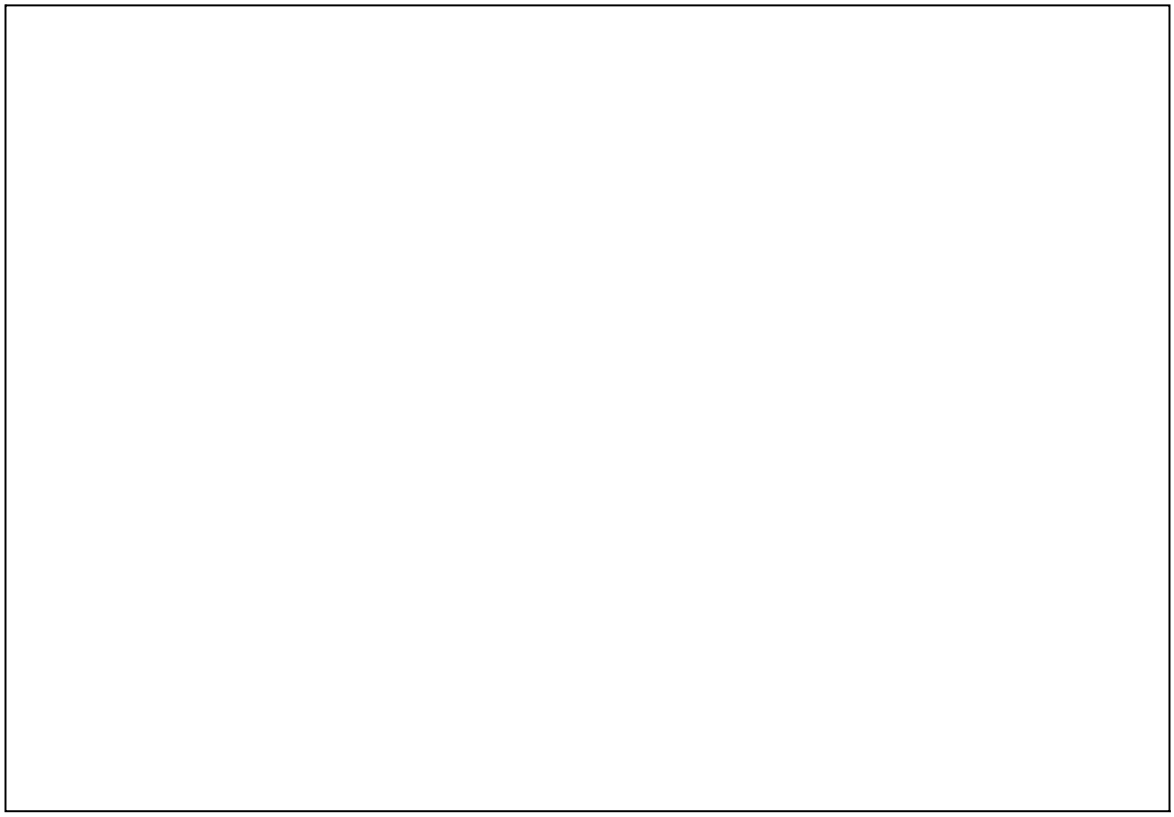


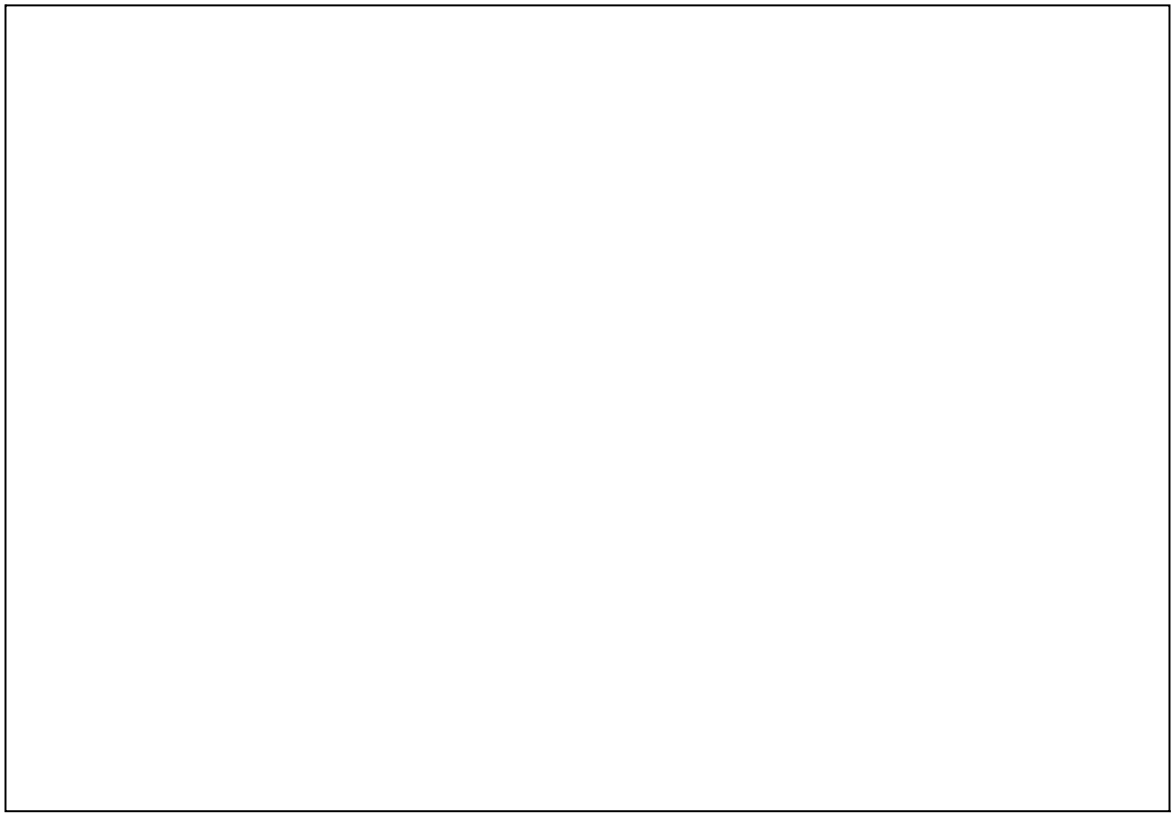


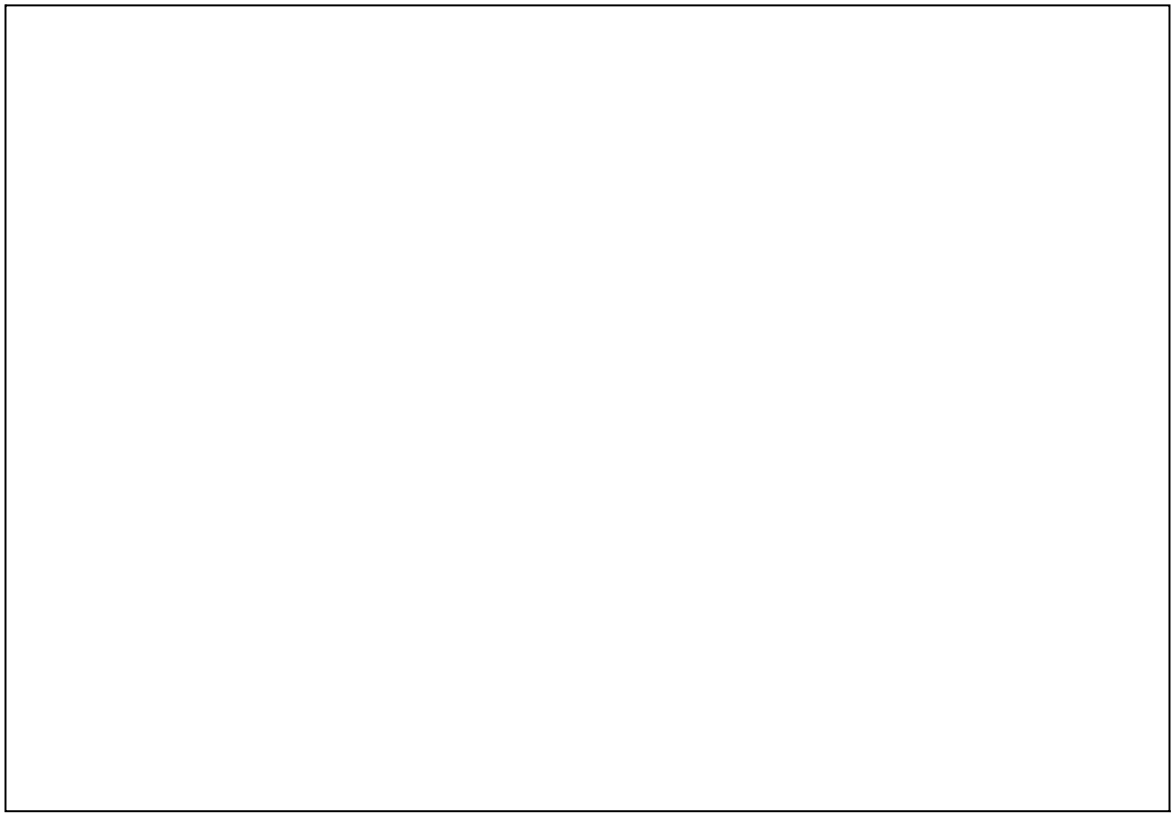


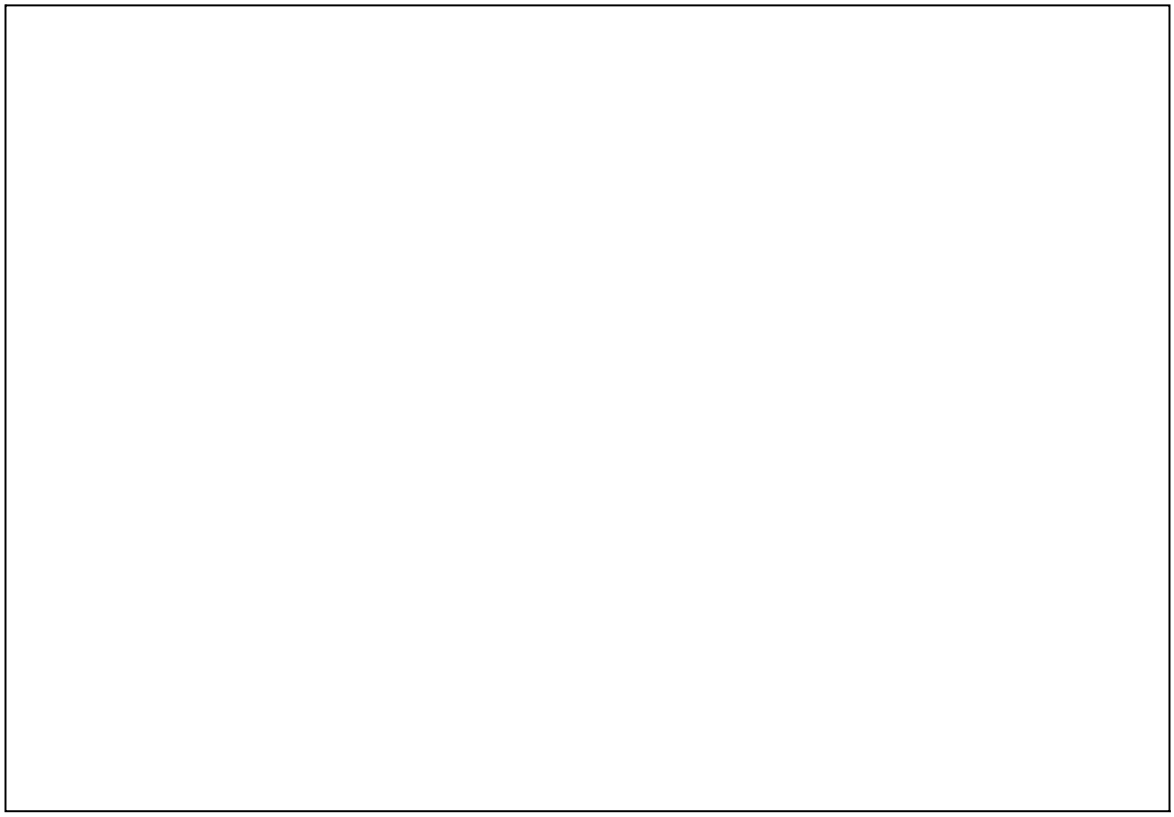


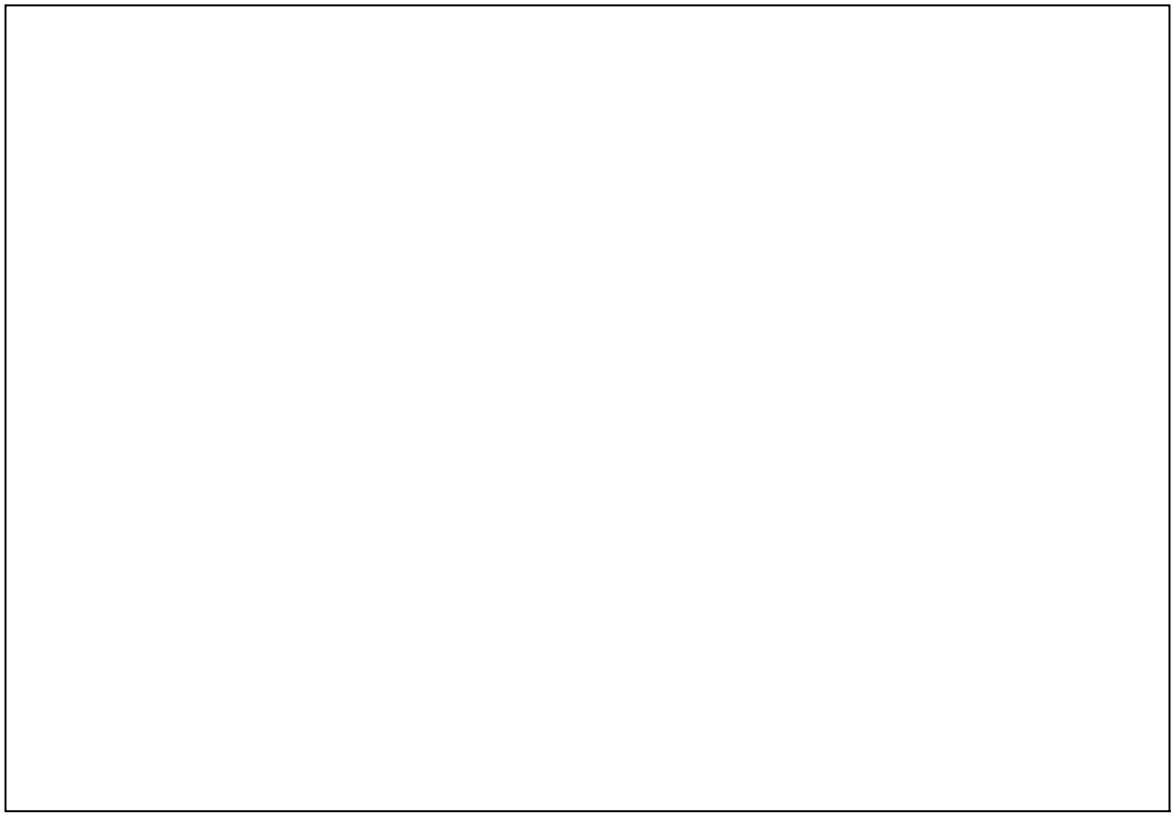


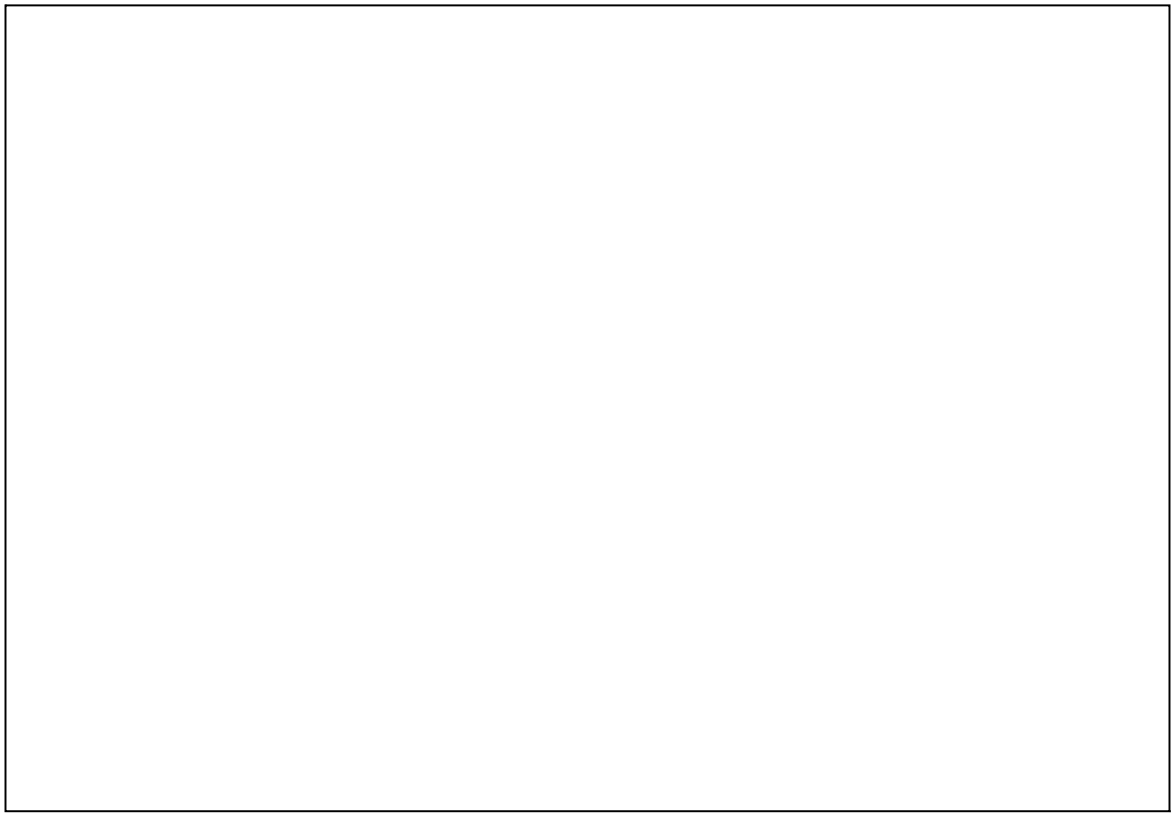


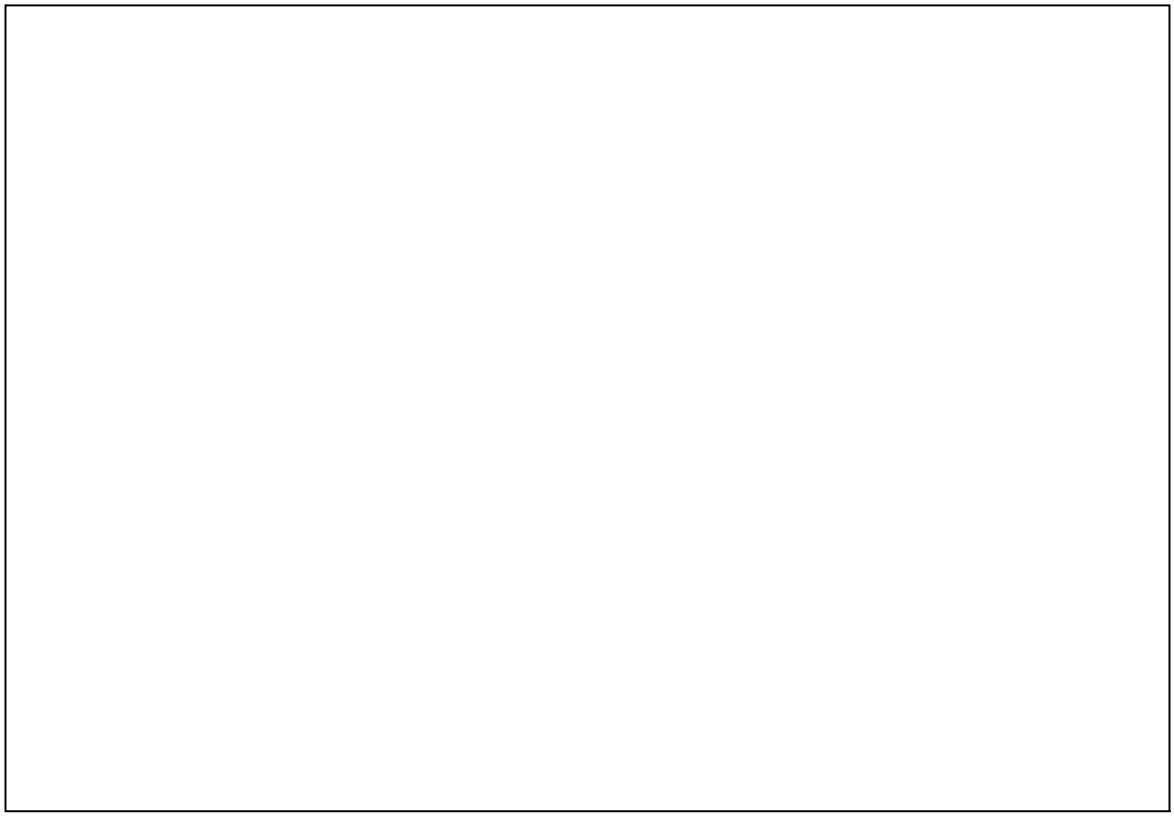


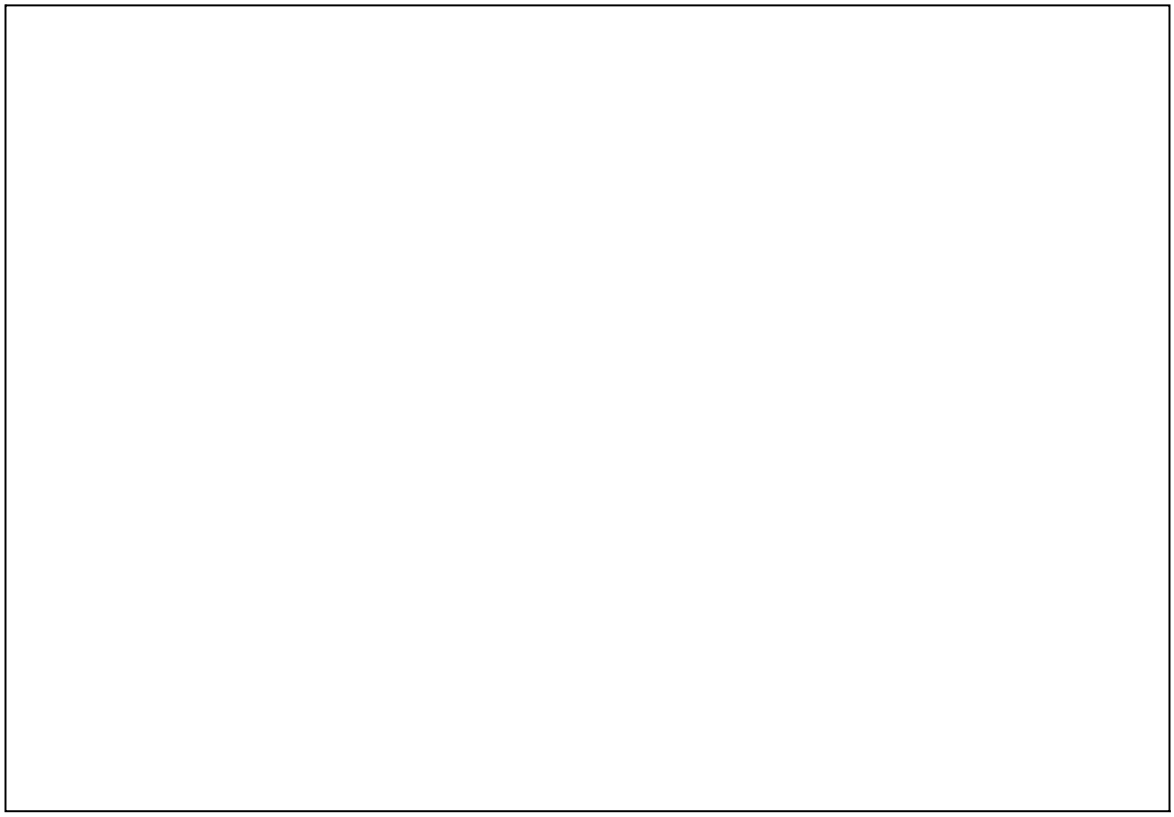


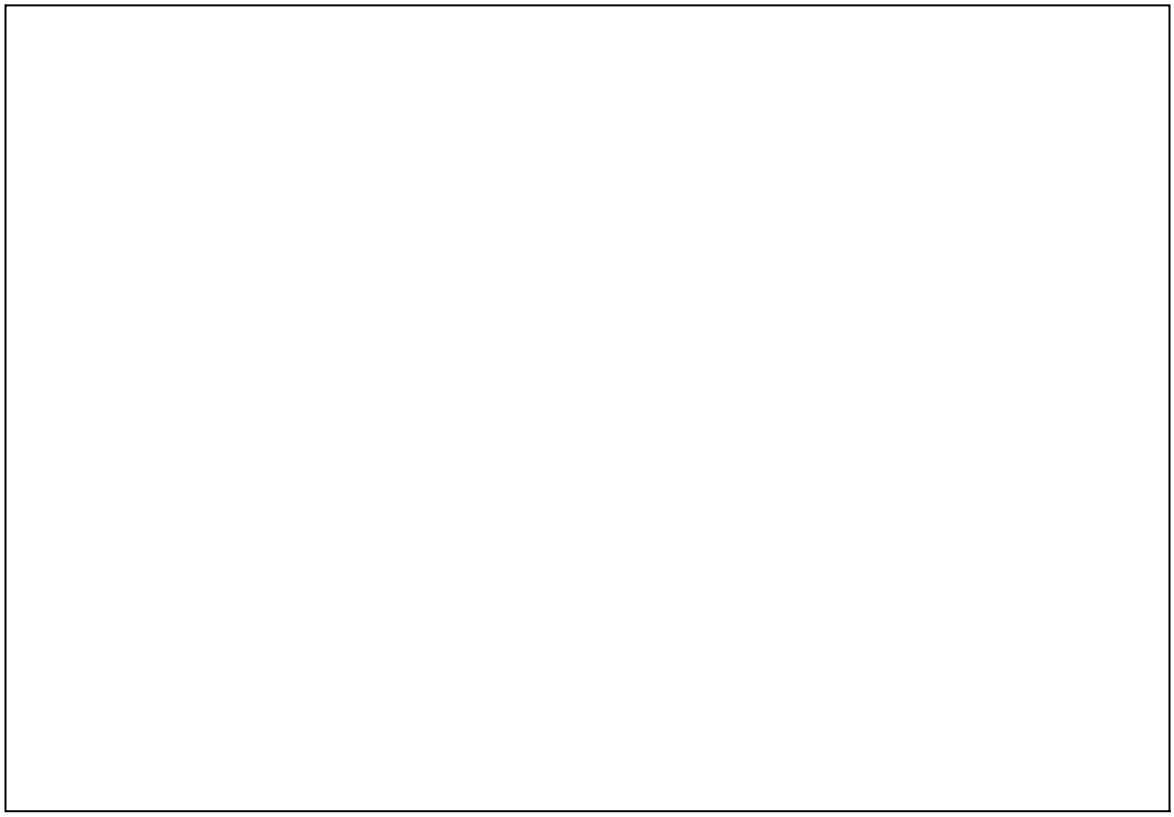


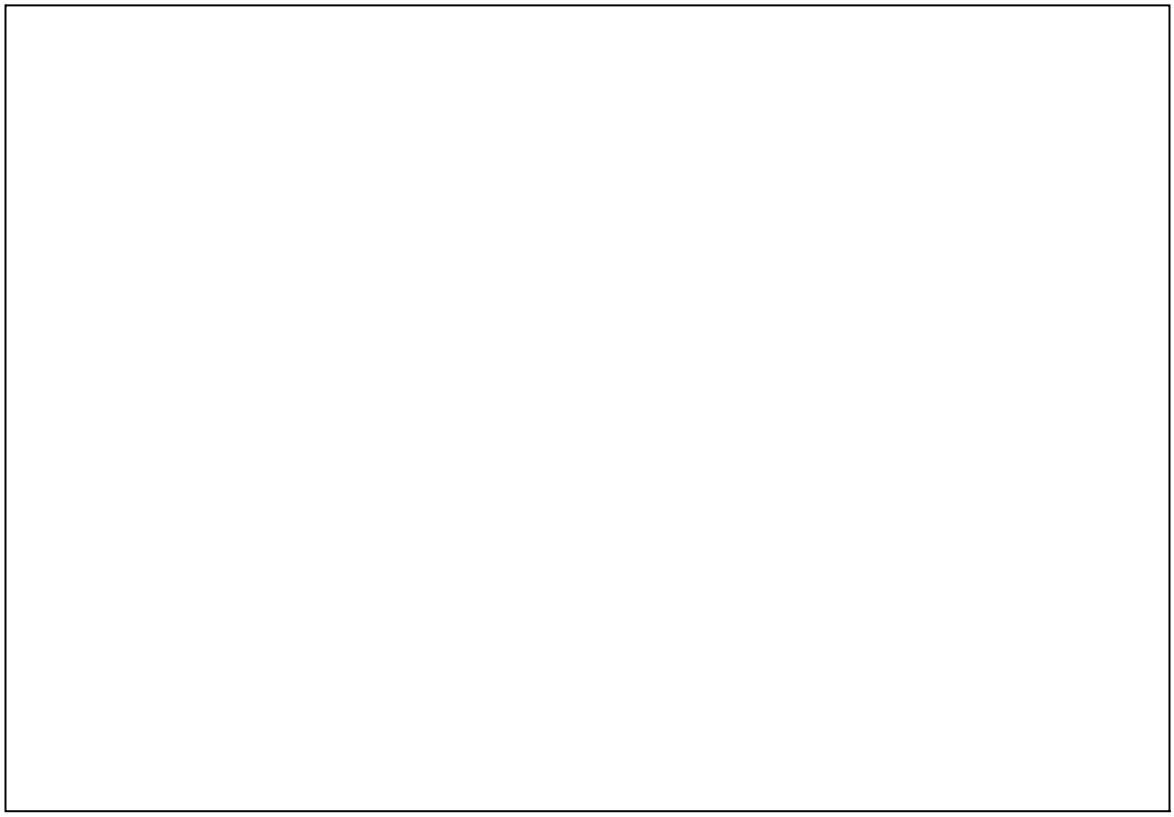


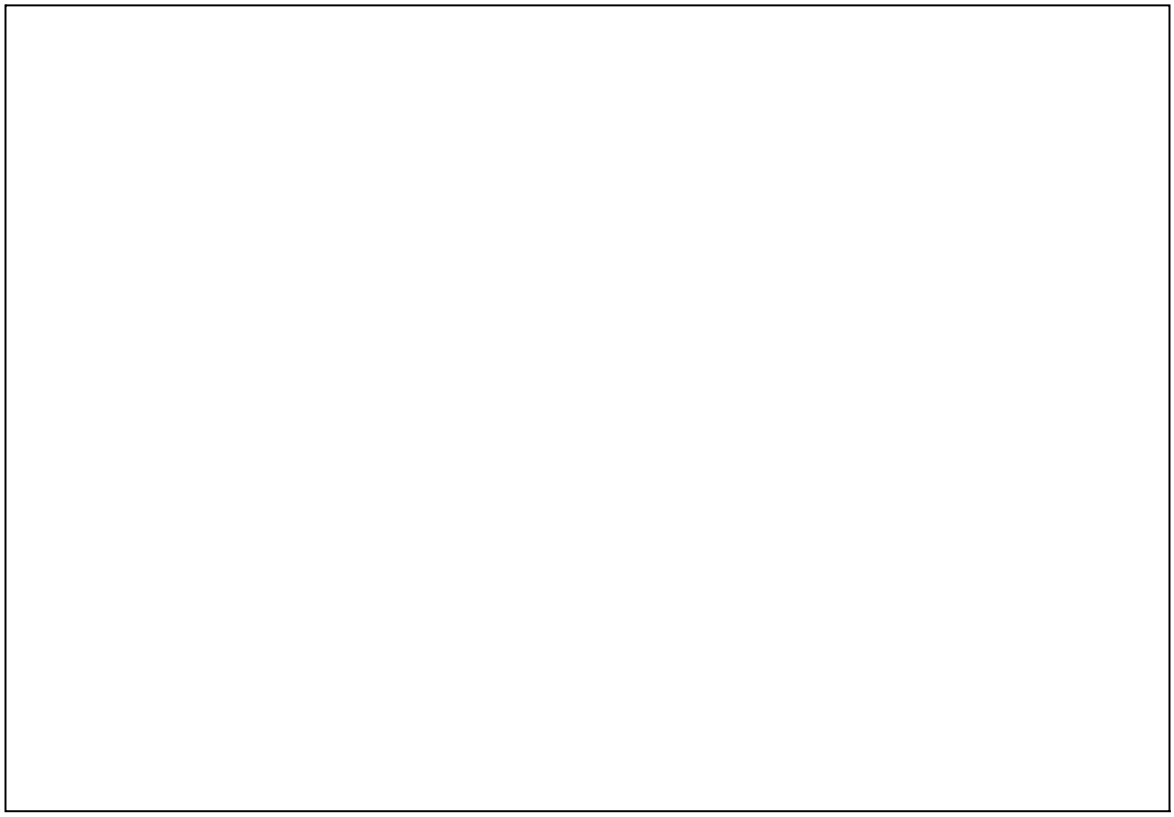


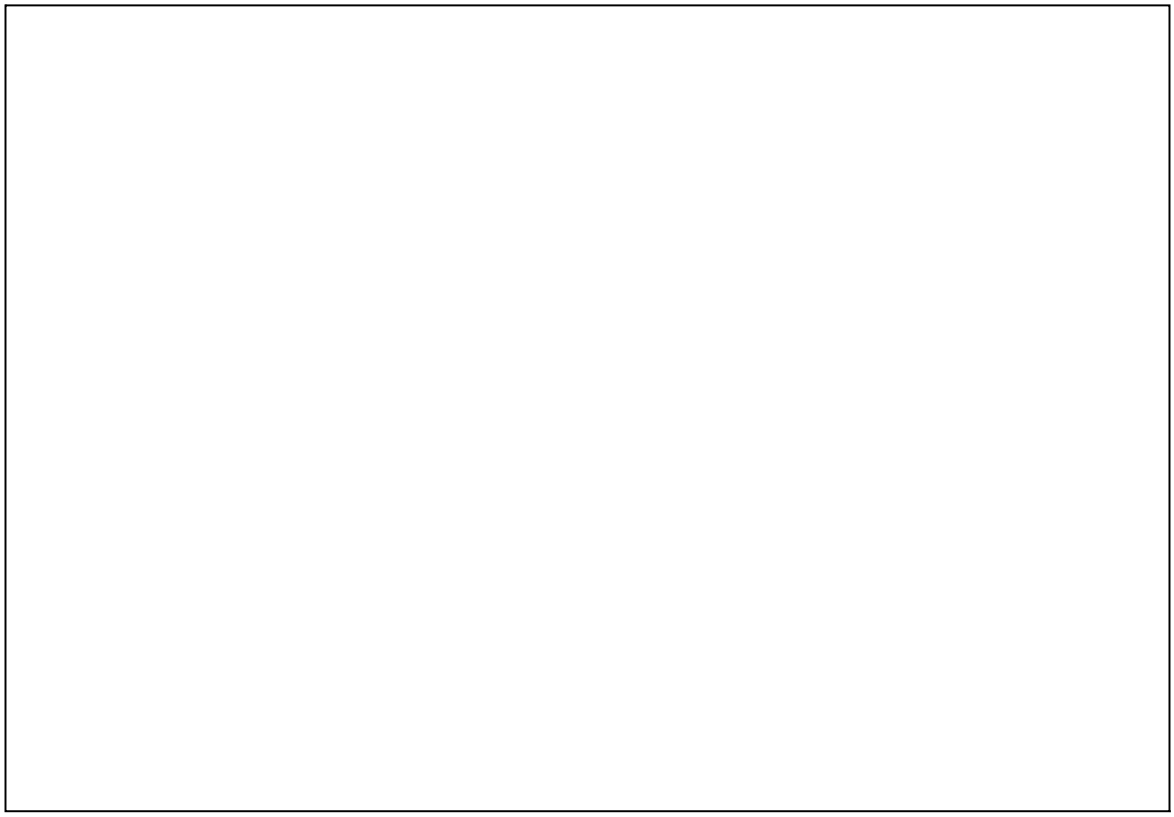


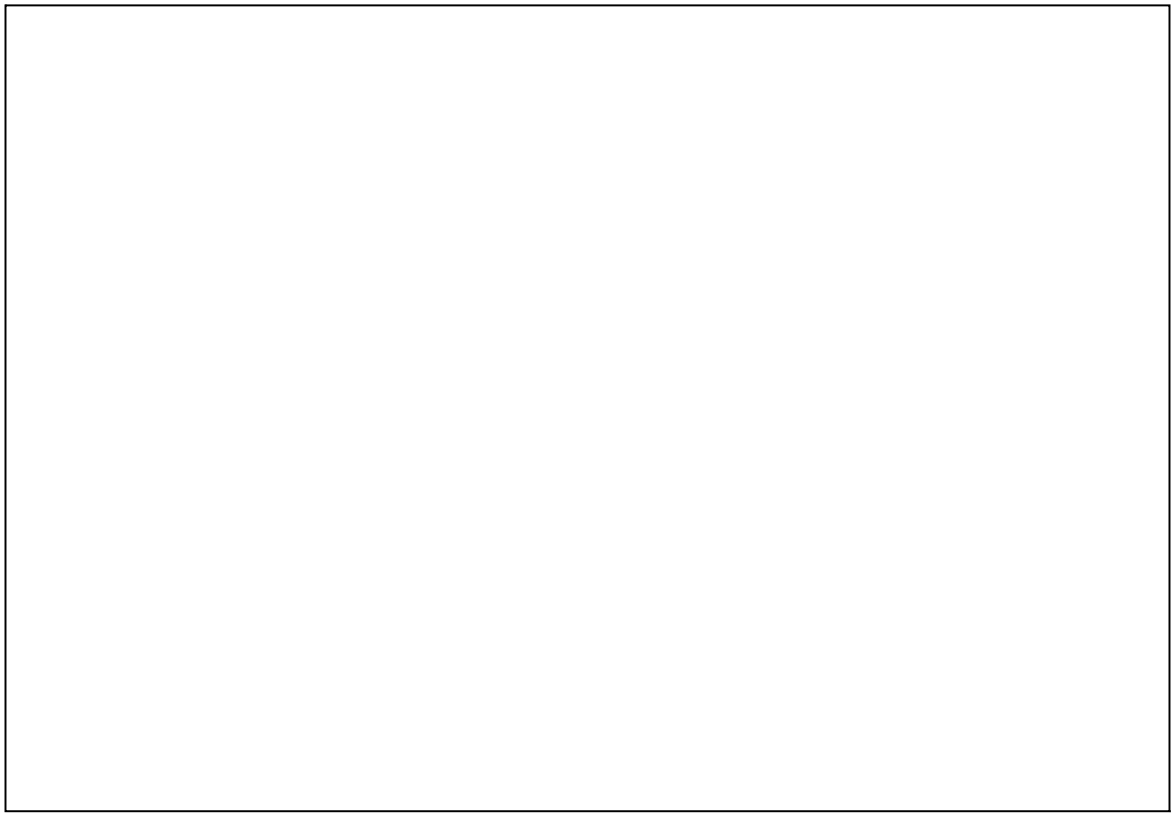


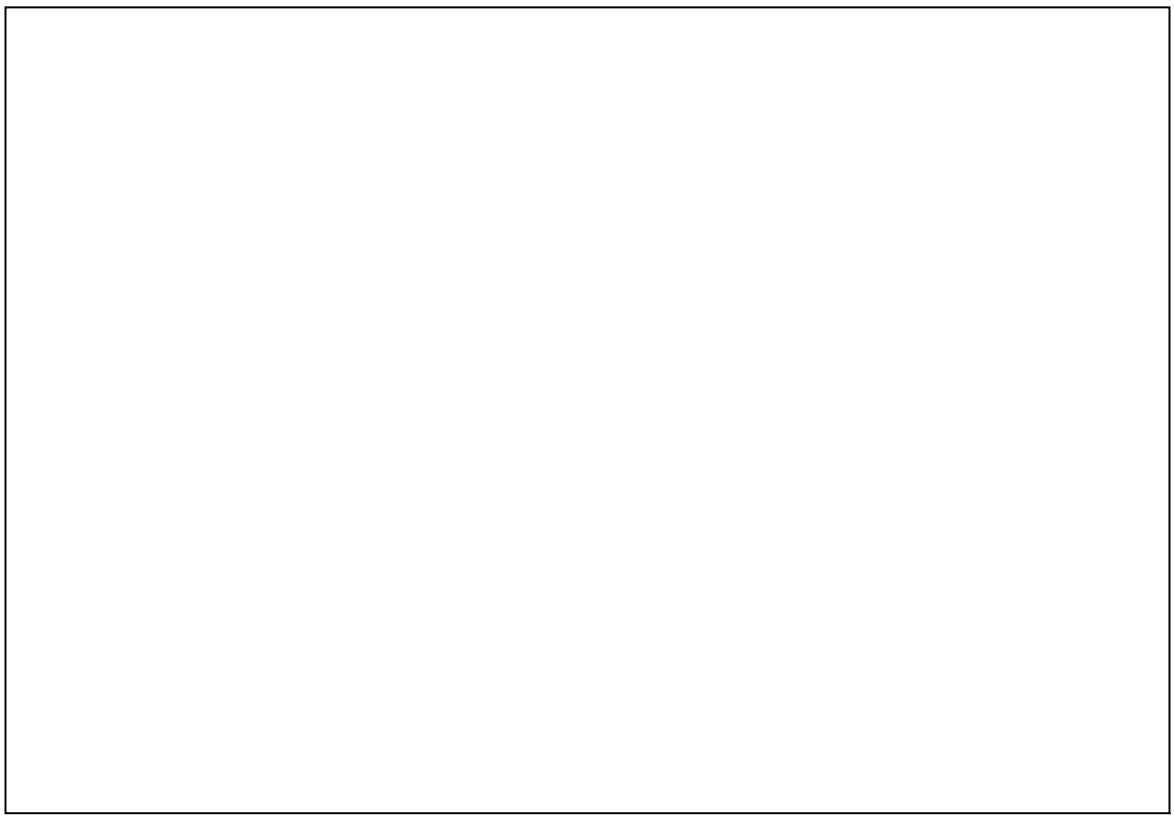












## Un peu de pratique

- Installer un vserveur qui fera office de serveur dns
- Installer un vserveur qui fera office de serveur dhcp
- Installer un vserveur qui fera office de serveur LAMP