Production

Maven build System and Automation, Deployment Options for Sakai

Dr Ian Boston
CTO,
CARET, University of Cambridge

Good morning, my name is Ian Boston, I am CTO for Caret, University of Cambrdige. Over the next 60 minutes I hope to give you an introduction to running Sakai in production.

A Technical Presentation

- Sakai in Production
- Build Systems
- Questions

Sakai in production is not just about the having a server up and running. A well run production environment will cover both the human aspects of deployment for example, acceptable user policy, and the technical infrastructure to run the installation and maintain it. I will introduce the planning for production and the build systems necessary to manage the chosen production instance.

Sakai In Production

- The Database
- App Servers
- Supporting Servers

Sakai, is a web application. It needs a number of servers to operate. Although it is possible to run Sakai on a single box, its worth thinking of the layers of athe architecture as distinct service components. Sakai uses a database server to store its data. The core application runs on one or more application servers and the application is supported by a number of other servers such as cluster wide shared file storage.

Supported Databases

- MySQL 4.1 with Query Cache Enabled.
- Oracle 10g with latest JDBC Drivers.

Sakai operates on a single database instance, and this is probably the first choice facing any one planning a deployment. Sakai is QA'd against 2 production databases, MySQL 4.1 with the query cache enabled, and Oracle 10g with the latest drivers. The choice of database is probably an institutional decision driven by the experience of the team and perhaps site wide licenses.

Database Server

- Definitely:
  - Don’t use HSQL in production.
- Probably:
  - Oracle scales further than MySQL
  - MySQL needs less maintenance
  - Oracle costs more, license and people.

So a quick run though of the things that should influence that choice. Firstly the binary demo package of Sakai runs with HSQL db out of the box. Don’t be tempted to use HSQLBD in production. It might work well for 10 users for a month, but it doesn’t scale at all, and has a slightly broken transaction model. In short, don’t use it in production.

Its probably true that Oracle will scale further than MySQL, although its really easy to find all sorts of statistics from each vendor on how much better their product is than the others. Its also probably true that MySQL is easier for the non expert to run. It works out the box, in general the installation process can be a few minutes long and for most applications the optimization and scaling is relatively simple.

Oracle on the other hand requires more resources to run in production. It certainly used to be the case that a production instance would require at least 1 day of experienced DBA per month to check the health of the DB and perform table-space maintenance. Sakai only addresses the table setup and does not get into table-space tuning. Its almost certain that you will need an Oracle DBA to perform the deployment and get the dynamic parameters on the tablespace correct.

And then there is the license.

Oracle licenses costs, usually per cpu, unless you have spent $M of bucks on one of their other applications and got site wide licensing arrangements.

So both MySQL and Oracle are valid production environments, and depending on you ultimate size you can make your own choice. Michigan with 40k students uses Oracle as does Indiana with 100k. Unisa has 180K students on MySQL.... so nothing is clear. At Cambridge we have a site wide license for Oracle, having bought Peoplesoft SIS and Oracle Financials, however we run on MySQL because we dont have dedicated Oracle DBA resource available.

Database Sizing

- Size the DB server correctly
- Choose a Machine with good I/O bandwidth
  - Memory
  - Disk

The main thing with the database is that you size it correctly. Its the center of the Sakai cluster and at the moment it is hit quite hard by the app servers. So start with the Hardware, chose something that can support enough memory, and has high I/O bandwidth to Memory and Disk. Today this probably means a well architected 64 bit server. If its going to be a big Oracle installation, it might be something special from Sun or IBM. Lower down the range there are plenty of x86 boxes that do the Job. At Cambridge we went for XServes here because we can throw memory into them and they have very good I/O pathways to both memory and I/O subsystems.
Hard pressed DB systems are driven by the I/O firstly to memory and then to Disk. Make certain that you get this right. Fast local disks are good, but also high bandwidth Disc subsystems. We are using 4GB FC connections to a San which offloads nearly all the I/O from the machine.

Database Resilience

- If the DB goes down, everything goes
- Disaster Prevention
- Disaster Recovery

So the DB is the center of the Sakai cluster and if it goes, everything goes. You need to think about how you are going to stop this happening either by putting in resilient hardware with redundant everything, or by putting in multiple machines in some sort of cluster.

Don't be tempted to go too far down the disaster mitigation strategies. The impact of major natural disaster on your VLE is probably not something that should force you to put you DB servers 50 miles apart.

Remember the app servers hit the DB hard and any latency you introduce between the app servers and the DB matters.
So, unless you have bags of Dark Fiber doing nothing, with sub ms packet latency, put your DB servers on the same local LAN as your App servers.

There there is the speed at which you can recover from a disaster.
If 10s downtime is Ok, then a hot db standby is probably Ok.
If 5 minutes a cold DB standby,
if 30 miniutes is Ok, you may be Ok with a cold recovery from backup.

The list goes on, work out the risk and find a solution that matches your risk profile.

Oracle - Disaster Prevention

- Real Application Cluster (RAC) - Hard for most to configure, can be costly, license + SAN
- Hot standby with hot redo logs.
- Resilient Hardware
- Database hardware virtualization

Looking at specifics.
As you would expect. Oracle has a whole range of solutions to disaster prevention. You could, if you have lots of experience time and money go for an Oracle RAC. They are hard to setup, they do need a SAN of some form and you will pay for each cpu in the cluster unless you can get a deal. The complexity of the RAC setup is probably a step too far for most Sakai installations.

Practical solutions to an Oracle cluster include a hot standby with redo logs, which can give fast switch over in the case of a failure.

You can take the resilience down to the hardware level with some resilient hardware solutions, that reduce the risk of hardware failure.

Or you can virtualize the OS.
Virtualization has the huge advantage of abstracting the hardware and done properly can give you migration of the virtual machine inside the data-center with interruptions as low as 100ms.

To virtualize Oracle well, you probably need to have a virtualization targeted at Oracle.

Oracle - Disaster Recovery

- Hot Standby, hot redo logs.
- backups
- virtualization migration to new hardware.

For recovery after the disaster, hot standby with redo logs is usually the chosen solution. It has the added advantage that backups can be taken from the hot standby machine to avoid downtime on the live server.

Obviously you need real backups of the DB, not forgetting that a disk backup of a running Oracle instance will almost never be valid. Its amazing how many times a sysadmin has don a cold backup of a hot service and wonders why it ends up in a strange state.

With visualization, you get the opportunity to snapshot the VM image. If you are using some sort of LVM under the VM image you get the opportunity to do this live which can give you options to backup the VM image any point in time, but don't forget that you will need to ensure your snapshot results in a valid oracle restoration point, just the same as cold backups of hot db's.

MySQL - Disaster Prevention

- MySQL Cluster - forget it - NDB has to fit in memory, and Sakai uses InnoDB
- HA with Master Slave replication - Failover easy - Failback hard.
- HA with DB on SAN - easyer.

MySQL has different options. First don't bother with the MySQL cluster, when you dig into the documentation on the NDB table type you find it has to exist entirely in memory and so is only really suitable for quite small databases. Coupled with the fact that Sakai is expecting InnoDB.

The main disaster prevention strategy for MySQL is a High Availability Cluster. There are 2 options for achieving this. Shared nothing with a slave database. The master is live and it replicates its redo logs to the slave which replays them. When the master goes, the slave gets reconfigured to be the master by the OS level HA switchover. Its relatively easy to make replication and failover work, but getting automated fail back is hard, as it requires all sorts of snapshots and redo log edits to get it to work.

Alternatively you can store the database on some sort of shared disk, and have a live DB instance with a cold standby. This is much easier to configure and the downtime from hardware failure can be as low as 5s if the configuration is good. Cambridge are running like this and its appears to work well with minimal effort.

MySQL - Disaster Recovery

- Master Slave replication.
- Backups
- Virtualization and VM migration, Xen, VMWare

MySQL recovery follows the same routes.

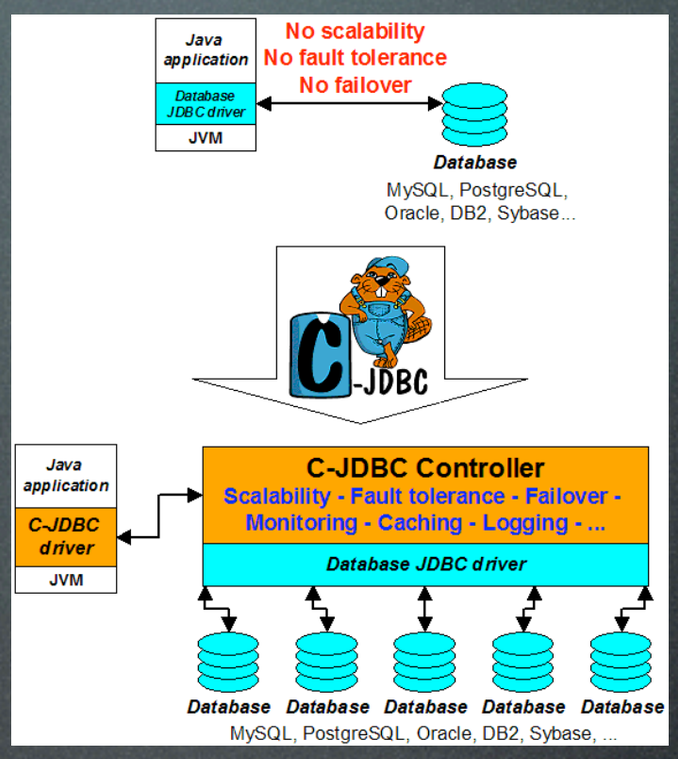
Master Slave replication gives you a read only instance. This can with work become the live instance quite rapidly. It also make it much easier to do backups from the slave as there is no interruption to the live master.

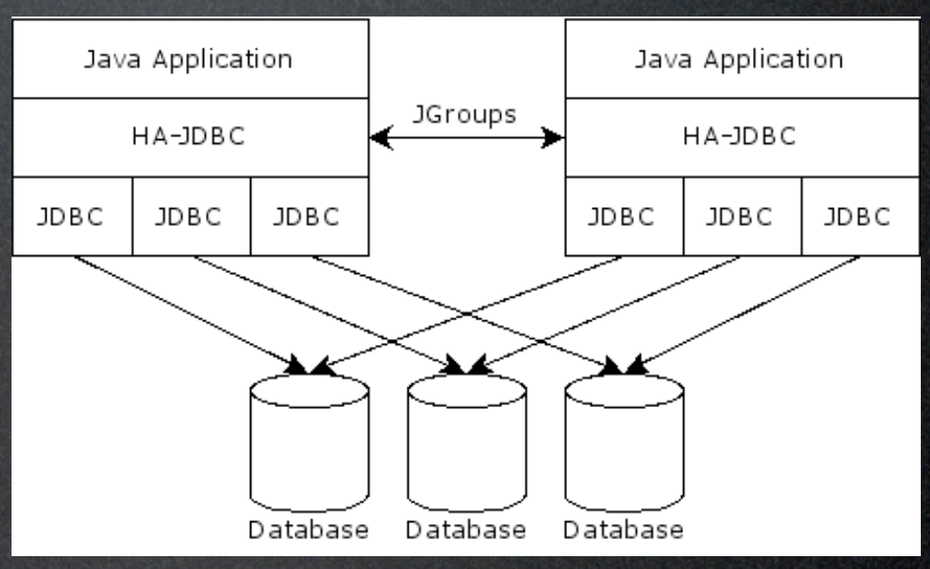
Backups for MySQL are relatively quick to recover from, but again cold backups of a hot database don't work.

And then there is virtualization. Since MySQL doesnt do anything special at the Hardware level, standard virtualization works just fine. Xen for instance claims <100ms migration of VMs in the same datacenter, so you could consider the Virtualization route as a way of making the DB resilient.

JDBC Solutions

- CJDBC
- HAJDBC





So Sakai doesn't support a clustered DB, but there are alternatives that you may have heard of. CJDBC and HAJDBC use some form of JDBC controller to maintain multiple copies of the same DB, and these look like possible alternatives.

JDBC Performance

- Writes Scale worse
- Reads Scale better

With these JDBC solutions, the writes are replicated over all instances and the reads are distributed, which make the reads faster and writes slower. This sounds good as there are many more reads in Sakai than writes.

Limitations

- “HA-JDBC does not safely support stored procedures that update sequences or insert rows containing identity columns.”
- CJDBC migt be the same.
- Probably wont work for Sakai..... but testing would tell.

However there are some issues that the App layer needs to be aware of. No triggers sorted procedures or anything that modifies state except directly as a result of the JDBC connection. The statement from HAJDBC is clear on its limitations and CJDBC might be same. Although this could work with Sakai, I know of a number of tools that use PK sequences and there may be some stored procedures to make oracle work quicker in key areas. So, you could try, but I feel that these probably wont work with Sakai.

Content

- File System
- Database - ok for Oracle, not for MySQL

One other thing to consider with the Database is where the bodies of files uploaded to Sakai are stored. Originally Sakai stored all content in the DB. In oralce you can dedicate table spaces to storing blobs and so this can be managed but it does create issues for the DBA's when performing backups. Putting the content in the DB for MySQL is a really bad thing, as access to those tables becomes really slow.

In general most sites are not storing all content on filesystem shared between the cluster nodes.

App Servers

- Tomcat 5.5.x, QA'd
- WebSphere port, IBM supported

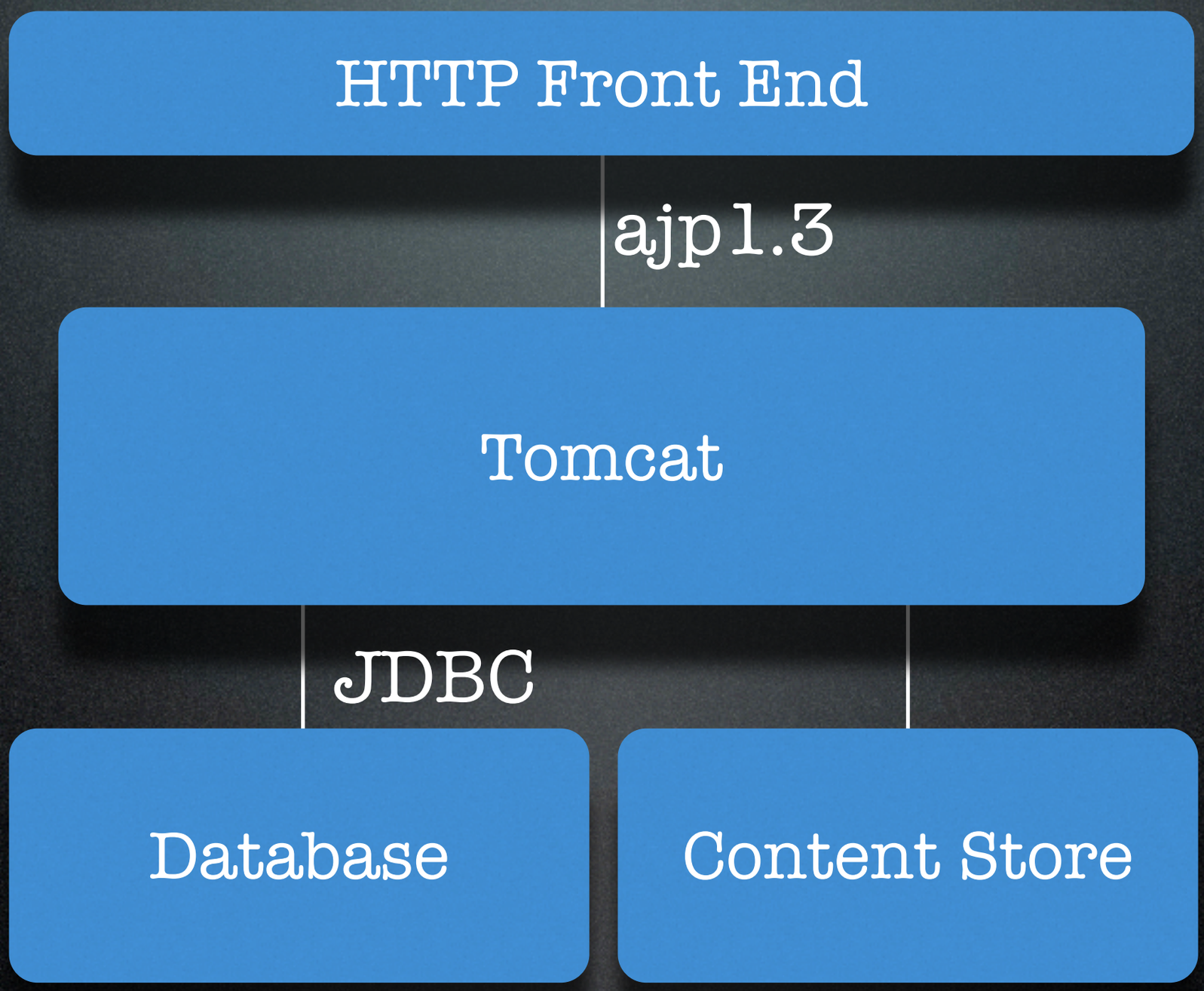
The Application server is at the core of Sakai. Its where all the java code runs. We QA against Tomcat 5.5 and will move to Tomcat 6 soon. There is a port of Sakai to IBM websphere and others are probably possible, WebLogic, JBoss, Jetty etc. There are only very minimal bindings to tomcat that can probably be recoded for other app servers.

Deployment Options

- Single Node
- Clustered
- High Availability

In deploying the app server you should think about the scale and reliability you need in this layer. The App servers can operate as a single node, or in a clustered environment and this gives the app server layer some level of high availability, however users are currently bound to each app server so they will notice if their instance goes down.

Single node



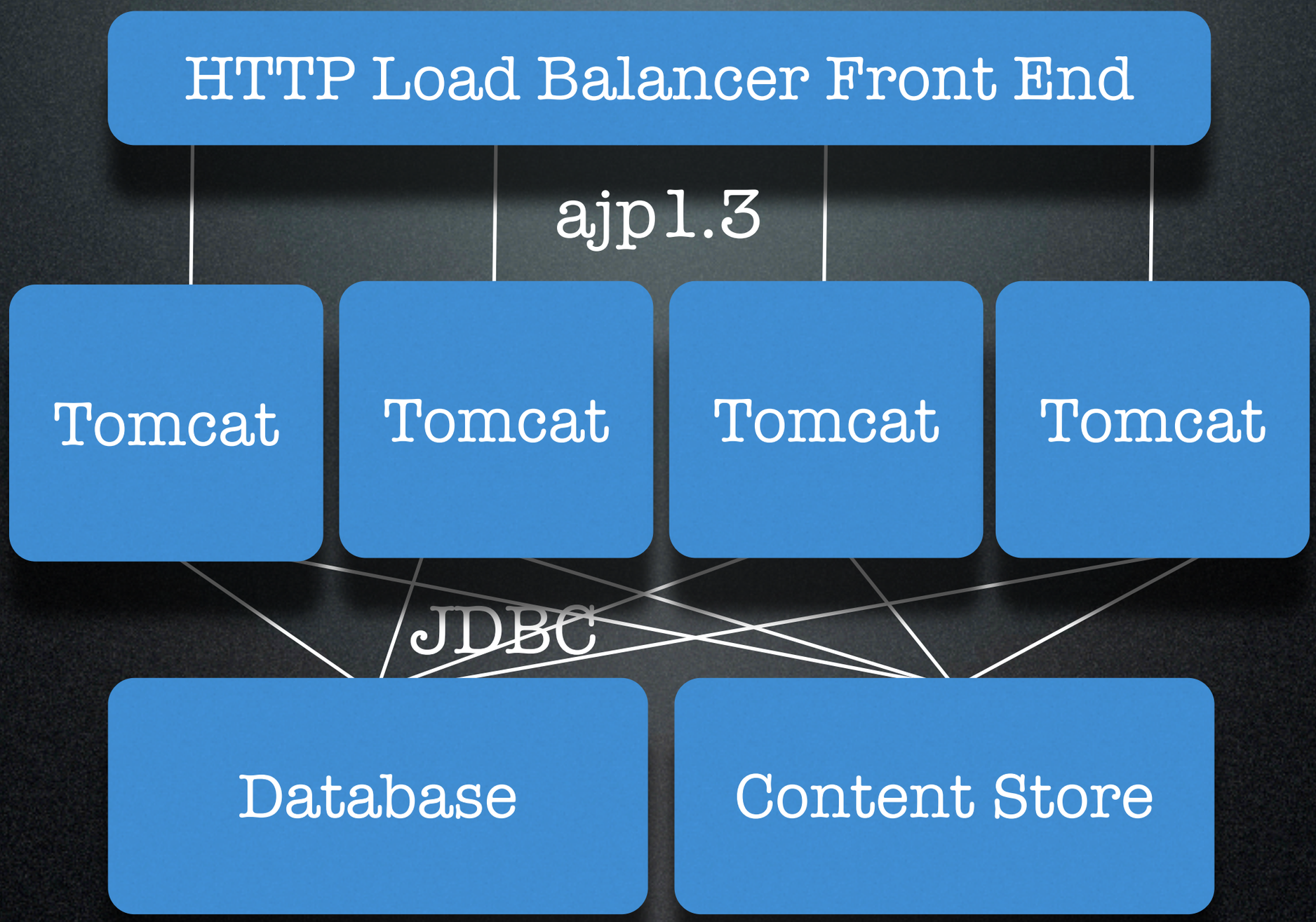
So this is what a single node deployment looks like. On the front end you almost certainly want something other than Tomcat to front up the Http requests, like Apache. Then Apache connects using AJP 1.3 to the tomcat instance that uses JDBC to talk to the database and accesses a content store for the bodies of files.

Single Node

- Possible to Run on One Node.
- upto 100 concurrent, database, appserver, everything.

Single node deployments, where everything the database, content store, front end and app server are all on the same machine, are probably good for up to 100 concurrent users. But if you think that you will see more than that, you really should consider separating out the components. Even if its to put the DB onto a different box.

Clustered Sakai



A clustered Sakai has more than one Tomcat. The cluster of Tomcat App servers are fronted up by an apache load balancer configured to have sticky sessions.

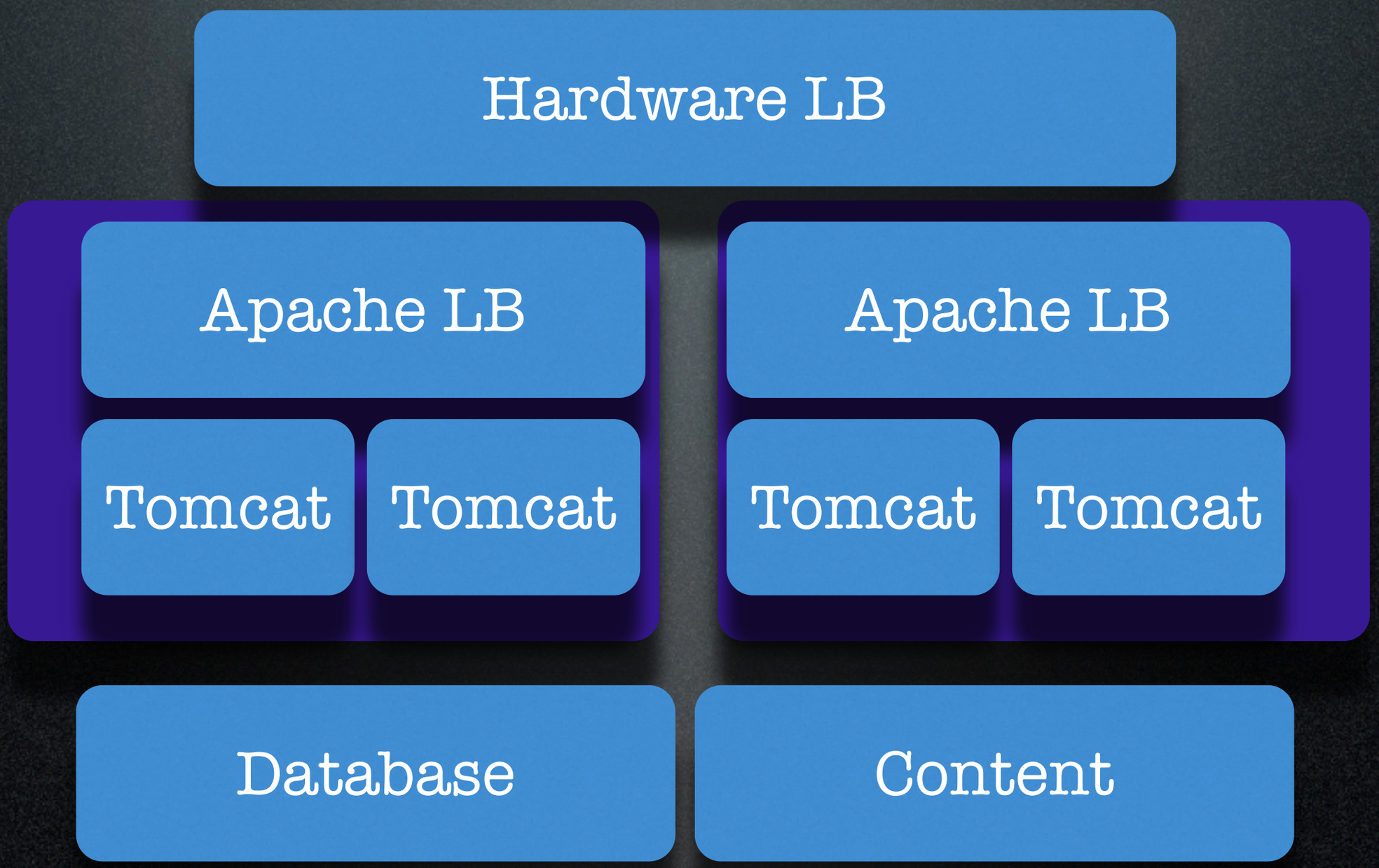
Cluster Considerations

- Increased load on DB Server, more connections.
- Load Balancer more complex
  - Big5 IP Load ballancer.
  - Zeus, ZTM
  - Apache, AJP13

But when you go to a cluster, there are things you need to think about. Many app servers surrounding a DB will hit the DB harder. there are additional queries associated with cluster maintenance that hit the DB. On the front end you need to start thinking about how IP traffic is distributed amongst the Tomcat app servers. THere are lots of options depending on expected load and budget. Hardware IP load balancers like Big5, or software solutions like Zeus ZTM Load balancer. But they all go down to something that talks AJP 1.3 eventually to send the traffic to Tomcat.

I think that unless you expect to see more than about 8M hits a day, a well configured apache instance with mod_proxy_ajp should be able to cope with the load. Thats only based on hosting other sites serving that order of pages.

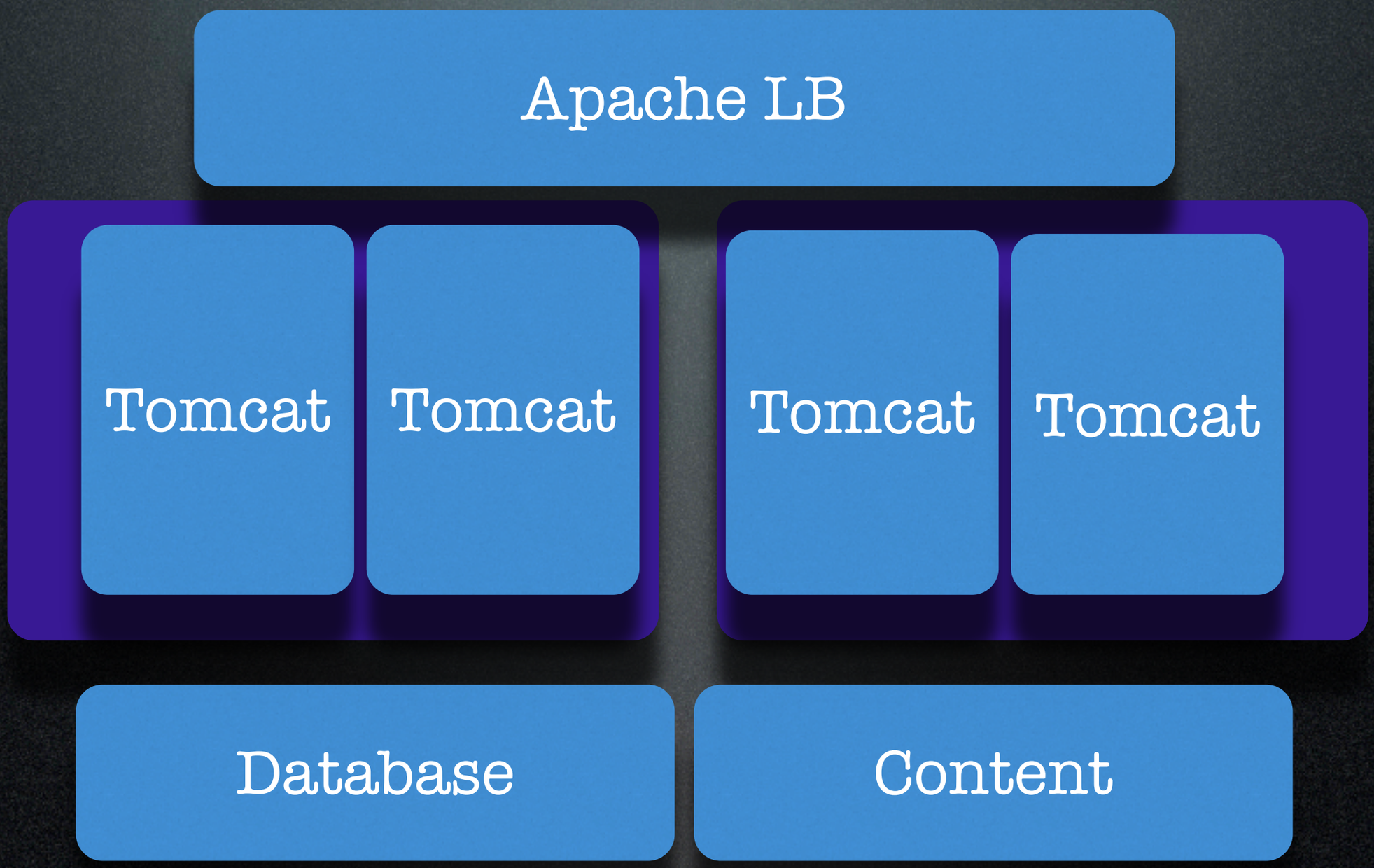
Hierarchical Structure



If you use a hardware LB the chances are that it will not be able to talk AJP1.3 so you will need to have an apache layer to manage the connection to tomat.

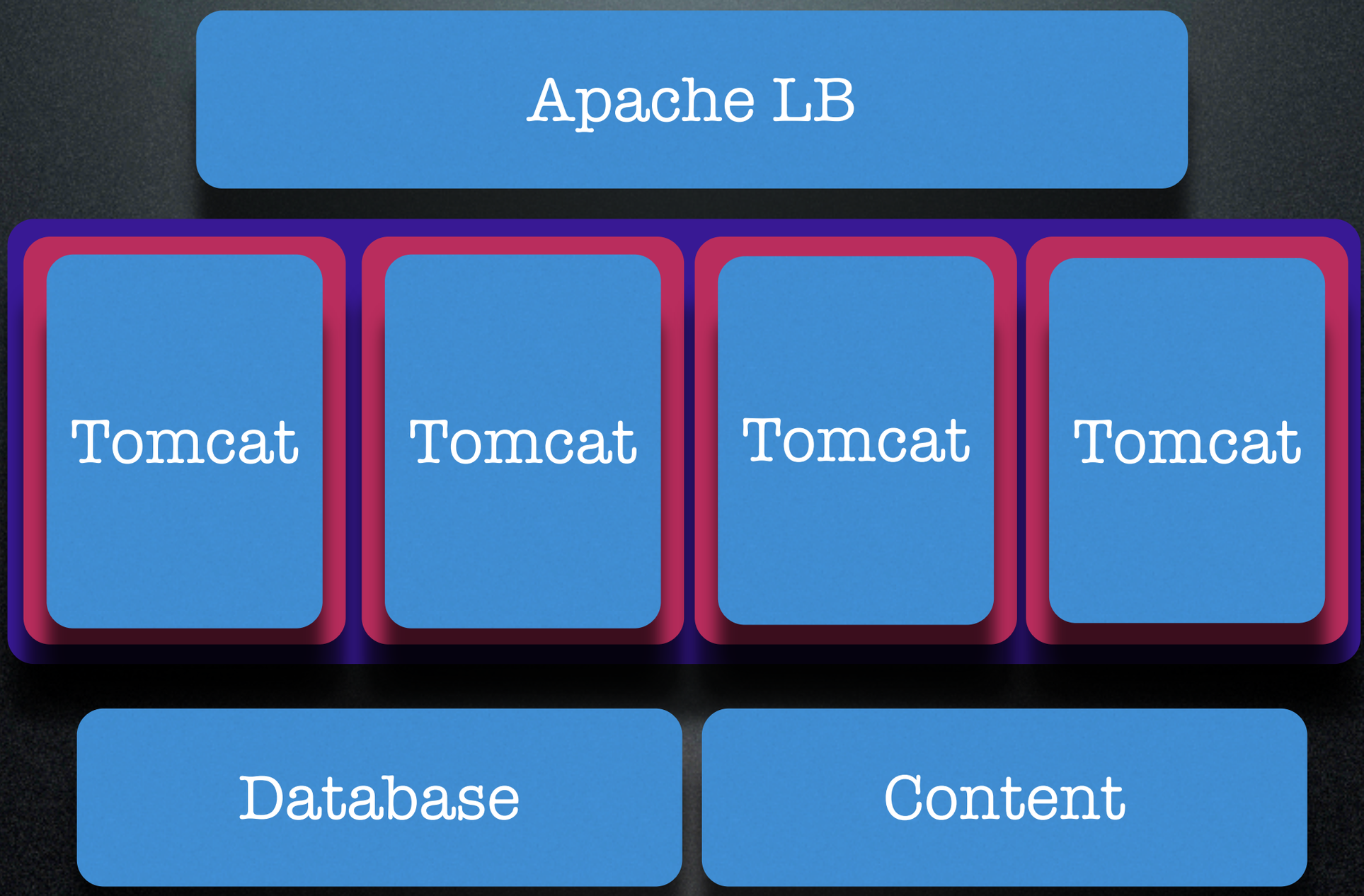
If you are going to use Java in 32 bit mode, and you have app server nodes with more than 2GB of memory, then you probably want to run more than one app server on each hardware node. This leads to a hierarchical structure, however it can be a pain to configure as each tomcat instance needs a different configuration to make certain there are not port clashes.

Hierarchical Structure



If you can use something that does talk direct to tomcat, then the installation becomes simpler, but you still need to make good use of the available memory on each hardware node.

Virtualize App Servers



One approach that works well to reduce the complexity of the app server deployment is to use virtualization at app server node layer. This way each virtualize app server has the same configuration as it believes its got an entire machine to itself.

Virtualized App Servers

- Apache AJP1.3 Load Balancer, HA Pair
- 2 App servers, with 4 Xen Virtual machines, 1 2G tomcat per Xen VM
- MySQL HA Pair, DB on SAN
- NFS Server HA Pair, content on SAN

At cambridge we take this approach. We use 2 hardware nodes with 8G of memory each running 4 Xen Virtual machines. So we have 8 app servers in total. We then have a HA apache front end, an HA MySQL back end and a HA NFS server for content bodies. All is running off a SAN.

Cambridge Hardware

- Dell 1U 2x Dual Core 3GHz Xeons. (8 vcpus per box)
- Apache, NFS share same HA pair 4GB
- App servers: 8GB, Xen 3.0, Debian Etch, 4 VM's per box.
- Database: Apple XServe 64Bit Intel Xeon 3Ghz DB pair, 16GB per box.

The hardware itself is mixed
Linux front ends,
Shared HA units
XServer for the Mysql ... ease of setup ... good IO bandwidth.

Others

- Indiana use a Virtualized Big Iron Box for 8 app servers. And a specialize Virtualized IBM box designed for Oracle Virtualization.
- Michigan use a big Oralce box and lots of individual app servers behind a Hardware LB.
- More information on confluence.

Others use different approaches

Indiana, heavily virtualized at tall layers with a Big5 IP LB

Michigan, real hardware, a big Oralce instance and lots of app servers on different subnets.

There is information of deployments on confluence.

Other Cluster configurations

- DDL, bring one node to build the DB, then turn off.
- LB need Sticky sessions.
- Rolling upgrades - possible but dont, Sticky session, schedule down time.
- Search Service, more complex.

Watch out for things in a cluster.

Sessions

Supporting Servers

- Shared File Storage, NFS
- Database Replication Slaves
- Load Balancer
- LDAP Cache

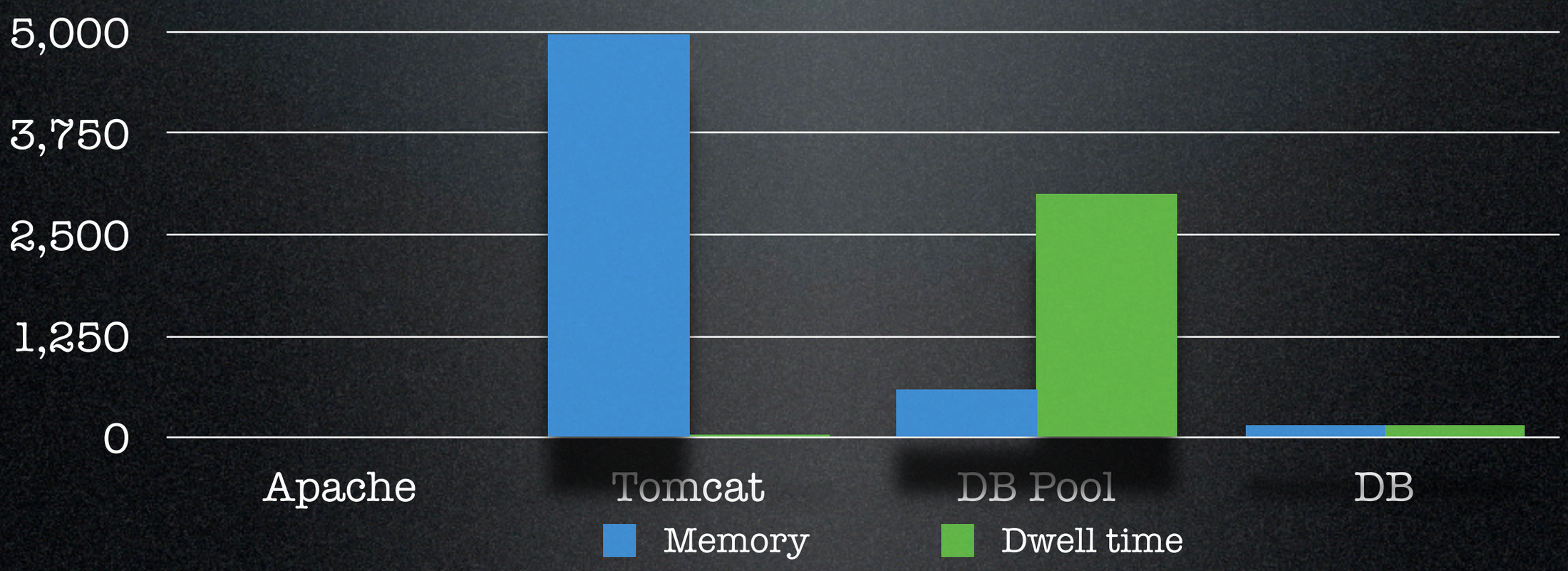
The core services have been talked about but you probably will have other services.

Tuning for Performance

- The Request Pipeline
- App Server
- Database

Blocked Pipeline

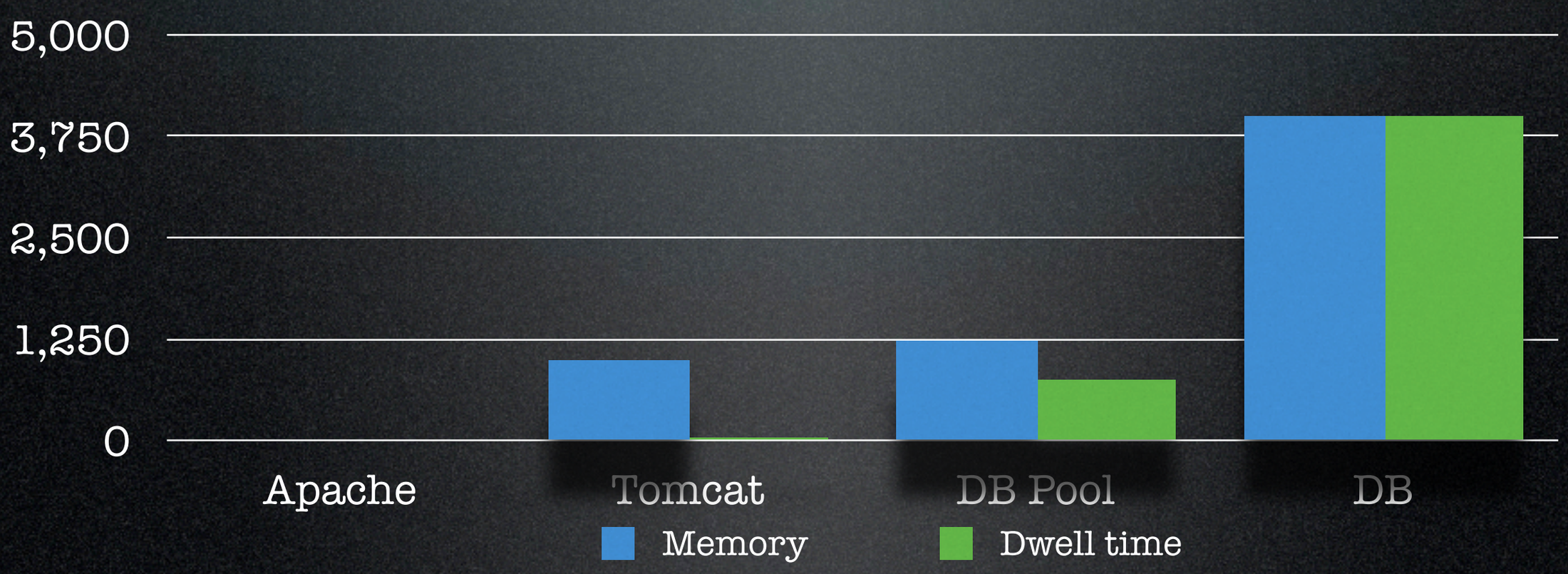
Apache 1000	Tomcat 1000	Tomcat DB Pool 50	DB 100



If the DB pools is too small request wait in the DB pools, consuming more memory than necessary, resulting in extra GC activity as the objects get of out the low cost eden part of the heap.

Overloaded Cluster

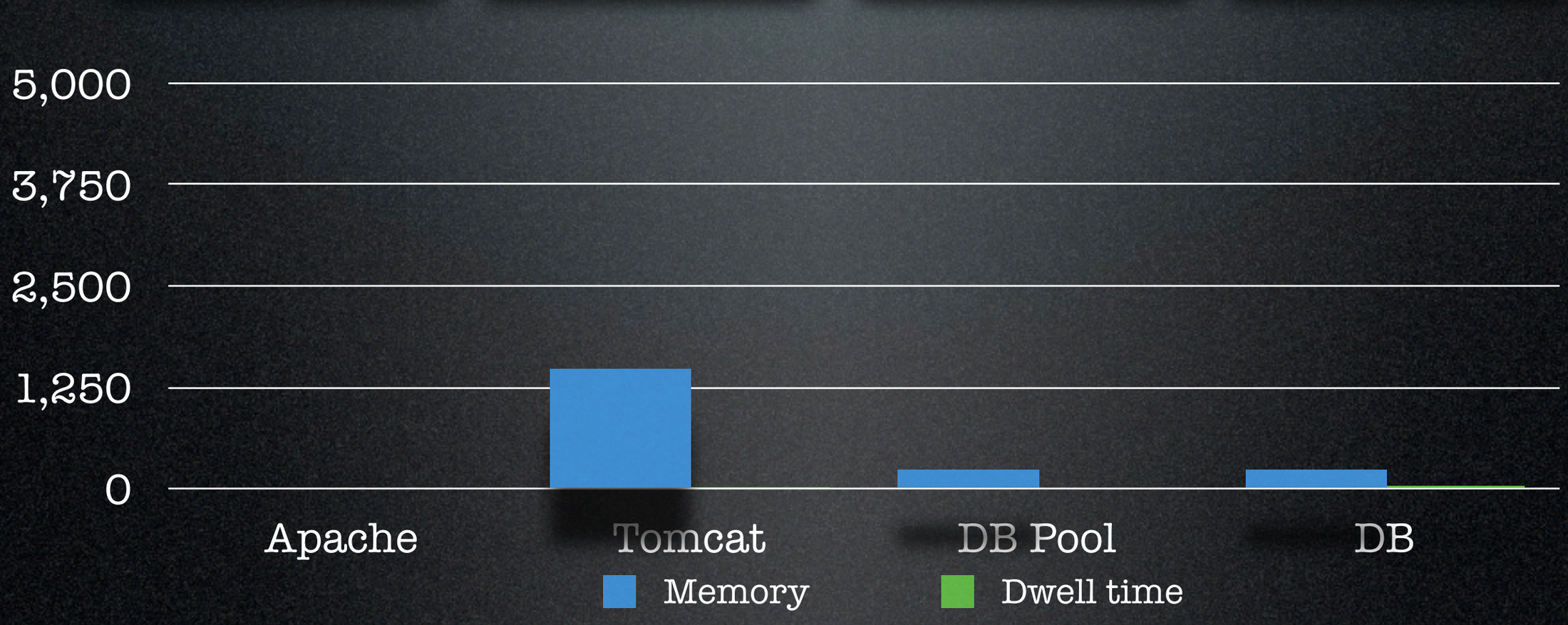
Apache 1000	Tomcat 1000	Tomcat DB Pool 250	DB 100



If the DB cant handle the load the connections have to wait until it can.

Tuned Pipeline

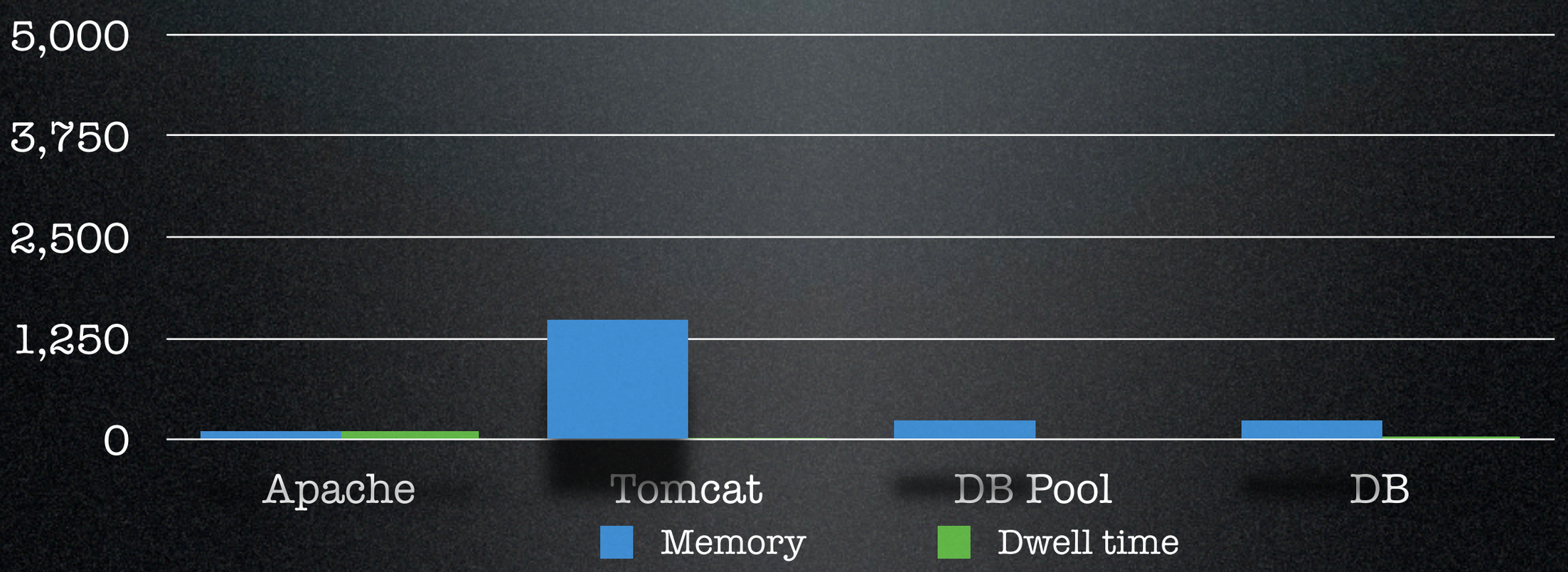
Apache 1000	Tomcat 250	Tomcat DB Pool 250	DB 250



A ballanced pipeline will process request with minimal waits so all the memory and all the cpu is consumed dealing with requests and less it used doing unnecessary GC operations.

Tuned Pipeline

Apache 1000, 5000 waiting	Tomcat 250	Tomcat DB Pool 250	DB 250



Then you can scale up apache, queing extra requests in apache where the cost is low. Apache can queue a request with lower impact than tomcat can

App server

- Heap:
  - Use JConsole
  - Use the throughput GC
  - eg “JAVA_OPTS=” -d64 -Xms2048m -Xmx2048m -XX:PermSize=256m -XX:MaxPermSize=512m -XX:NewSize=384m -XX:MaxNewSize=512m -XX:SurvivorRatio=16 -XX:+UseConcMarkSweepGC -XX:+UseAdaptiveSizePolicy ”
  - Ask
- DB Pool.
  - Oracle: Don’t create or destroy connections, expensive.

When running the app server look at how its running with JConsole or something similar, tune the java opts to make efficient use of the heap.

32 bit or 64bit

- Should be able to run in 32bit.
- Sad fact is, 2.4 started to need 64bit before patching.
- Longer term, we should only need 64bit to handle greater load, not just to survive.

Database Tuning

- Oracle:
  - Read a Good book on performance tuning for Oracle..... and DO IT, SGA, Partitions, IO, etc etc, regularly
- MySQL
  - Size the DB and turn the Query Cache on

Build Systems

- Source Configation Mangement
- Build systems
- Targets
- Localizations
- Automation

SCM

- Subversion Source Repository
- Trunk - Fast moving not for production
- Tags - The release
- Branches - The release with patches
- Take the Tag, but watch for patches and think about switching to branch.

Build configuration

- Have your own base SVN directory
- In 2.5 have your own pom.xml (more later)
- Configure svn:externals to load the modules and versions you want to build
- Patch overlay before build

SVN Setup

Revision 5530: /projects/camtools

- ..
- branches/
- tags/
- trunk/

Powered by Subversion version 1.4.3 (r23084).

search	https://source.sakaiproject.org/svn/search/branches/sakai_2-4-x
sections	https://source.sakaiproject.org/svn/sections/branches/sakai_2-4-x
site	https://source.sakaiproject.org/svn/site/branches/sakai_2-4-x
site-manage	https://source.sakaiproject.org/svn/site-manage/branches/sakai_2-4-x
syllabus	https://source.sakaiproject.org/svn/syllabus/branches/sakai_2-4-x
test-harness	https://source.sakaiproject.org/svn/test-harness/branches/sakai_2-4-x
textarea	https://source.sakaiproject.org/svn/textarea/branches/sakai_2-4-x
tool	https://source.sakaiproject.org/svn/tool/branches/sakai_2-4-x
user	https://source.sakaiproject.org/svn/user/branches/sakai_2-4-x
util	https://source.sakaiproject.org/svn/util/branches/sakai_2-4-x
velocity	https://source.sakaiproject.org/svn/velocity/branches/sakai_2-4-x
web	https://source.sakaiproject.org/svn/web/branches/sakai_2-4-x
webservices	https://source.sakaiproject.org/svn/webservices/branches/sakai_2-4-x
mailtool	https://source.sakaiproject.org/svn/mailtool/branches/sakai_2-4-x
usermembership	https://source.sakaiproject.org/svn/usermembership/branches/sakai_2-4-x
polls	https://source.sakaiproject.org/svn/polls/branches/sakai_2-4-x

Maven Build

- 2.4.x builds with Maven 1
- Post 2.4 is Maven 2

Maven 1

- Setup Maven 1
  - Download, Install see http://maven.apache.org/maven-1.x/start/install.html
  - configure build.properties
    maven.repo.remote = http://source.sakaiproject.org/maven/
    maven.tomcat.home = /Users/ieb/Caret/sakai22/tomcat/
  - maven plugin:download -DgroupId=sakaiproject -DartifactId=sakai -Dversion=2.2
  - setup MAVEN_OPTS
    export MAVEN_OPTS= -Xms168m -Xmx512m -XX:PermSize=24m -XX:NewSize=64m
- Write a build script
  - eg svn co https://saffron.caret.cam.ac.uk/svn/project/camtools/tags/2.4.x
    sh patch-and-overlay.sh
    maven pack-demo
  - Pack Demo produces a tarball of the sakai image containing tomcat ready to deploy.

Maven 1 Structure

- Projects built by the Maven 1 Multiproject reactor
  - Scans the disk for project.xml
  - Works out a dependency graph
  - builds according to dependency graph.
- master/project.xml, master/project.properties
  - Defines the properties of common jars.
  - All project.xml's extend master/project.xml
- Sakai Plugin
  - Responsible for Deployment into shared/lib common/lib components and webapps

Maven 2

- Dependency Management, maven 1 has no dependency management
- Faster repository
- More complete configuration.
- /master/pom.xml is the base
- /pom.xml extends /master/pom.xml
- /project/pom.xml extends /pom.xml
- project poms extend /project/pom.xml

Maven 2 installation

- Maven 2.0.6 or later
- Download, unpack, set JAVA_HOME, add maven-2.0.7/bin to path.
- mvn --version

Maven 2 build

- No configuration, all in pom.xml's
- mvn -Ppack-demo install
- tarball in /pack-demo
- Developer builds
- mvn clean install
- mvn sakai:deploy -Dmaven.tomcat.home =/opt/tomcat

Customizing the build

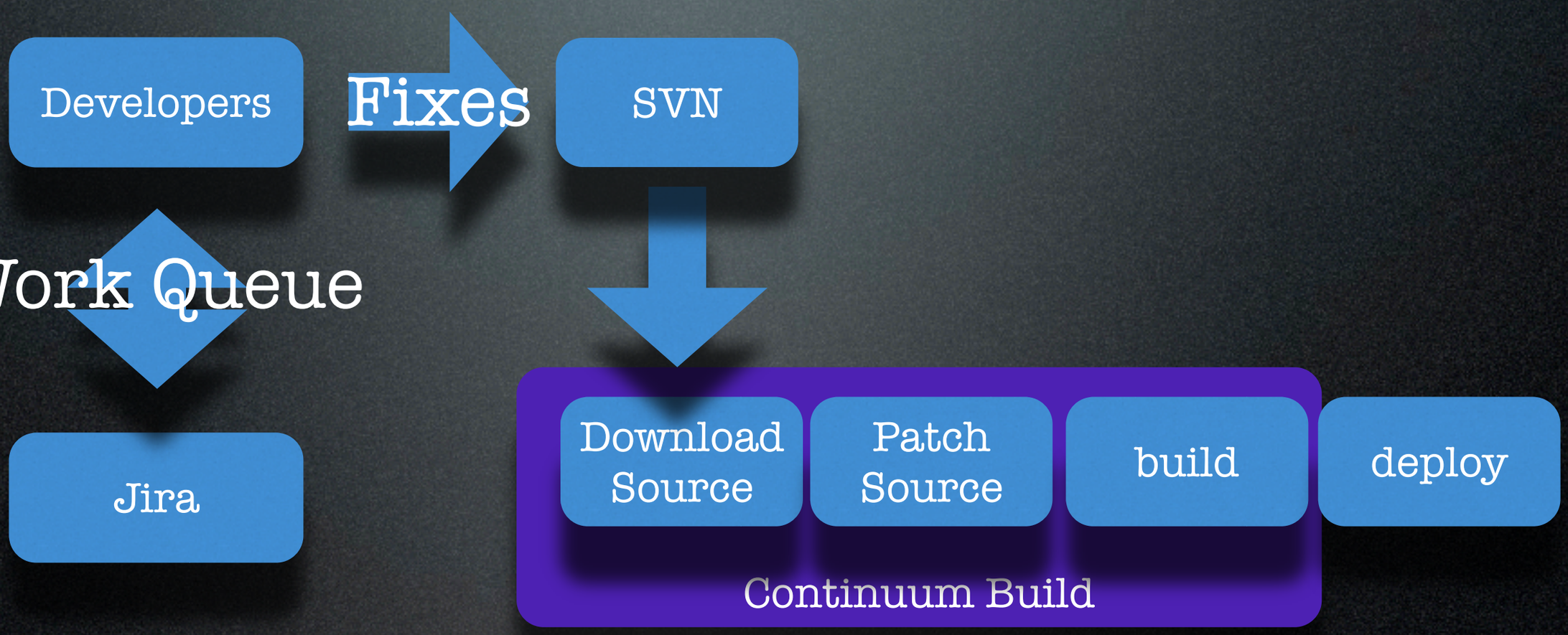
- Change the base pom.xml modules
- use maven 2 profiles

```
<profiles>
  <profile>
    <id>pack-demo</id>
    <modules>
      <module>pack-demo</module>
    </modules>
  </profile>
  <profile>
    <id>mini</id>
    <modules>
      <module>access</module>
      <module>alias/alias-api/api</module>
      <module>alias/alias-impl/impl</module>
      <module>alias/alias-impl/pack</module>
...
      <module>velocity</module>
      <module>reset-pass</module>
    </modules>
  </profile>
  <profile>
    <id>full</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <modules>
      <module>master</module>
      <module>access</module>
      <module>alias</module>
...
      <module>reset-pass</module>
    </modules>
  </profile>
</profiles>
```

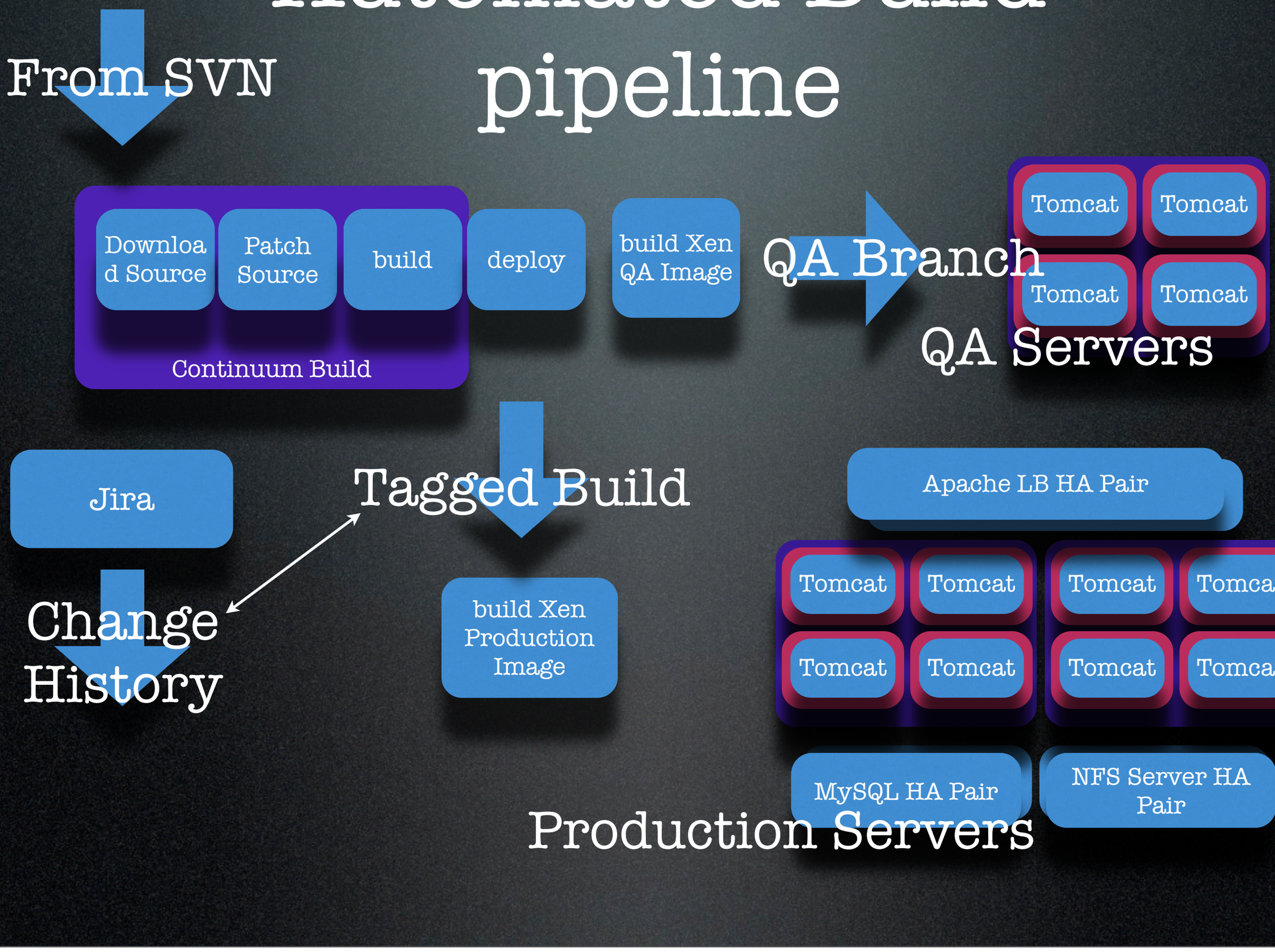
Targets - Options

- dont build take the binary, ok for 100% standard deploys
- developer build to to a app server overlay
- pack demo get a tomcat/sakai tagball

Automated Build pipeline



Automated Build pipeline



Thank you and Questions

Finally, thank you, and are there any questions.