Platform™

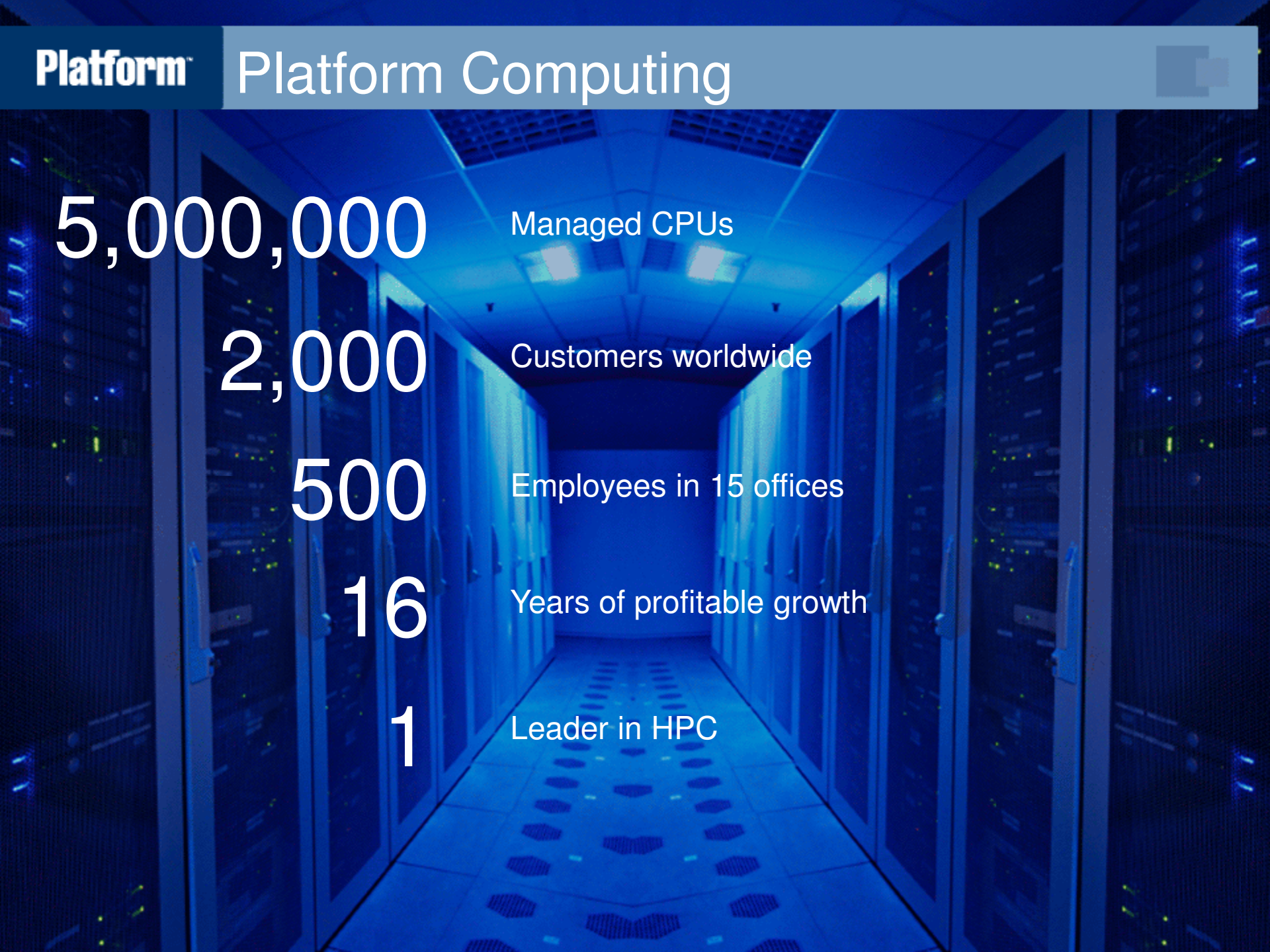
The Power of Sharing

Scheduling for mixed parallel and sequential workload

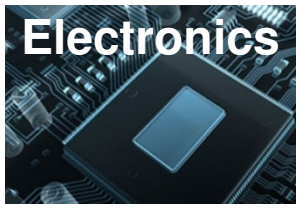
Robert Lurin, Owen Fraser-Green, Bernhard Schott
Platform Computing

Agenda

- Introduction
- Maximizing the efficiency on multicpu/multicore machines : multicore support
- How to guarantee that parallel jobs can run in a mixed sequential/parallel environment
- Maximizing the efficiency of memory usage :
  - exclusive user scheduling
  - memory bandwith scheduling
- Taking in account the topology in a compute farm
- Variable-size parallel jobs
- Maximizing the efficiency for very short jobs : transient sub-scheduling
- Application profiles
- Coupling Batch scheduling with virtualization

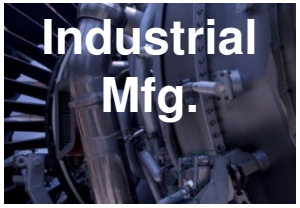

Platform
Platform Computing
5,000,000
Managed CPUs
2,000
Customers worldwide
500
Employees in 15 offices
16
Years of profitable growth
1
Leader in HPC

Platform Industries Served by Platform



- AMD
- ARM
- Broadcom
- Cadence
- Cisco
- Infineon
- MediaTek
- Motorola
- NVidia
- Qualcomm
- Samsung
- Sony
- ST Micro
- Synopsys
- TI
- Toshiba

- BNP
- Citigroup
- Fortis
- HSBC
- KBC Financial
- JPMC
- Lehman Brothers
- LBBW
- Mass Mutual
- MUFG
- Nomura
- Prudential
- Sal. Oppenheim
- Société Générale



- Airbus
- BAE Systems
- Boeing
- Bombardier
- Deere & Company
- Ericsson
- Honda
- General Electric
- General Motors
- Goodrich
- Lockheed Martin
- Nissan
- Northrop Grumman
- Pratt & Whitney
- Toyota
- Volkswagen

- Agip
- BP
- British Gas
- China Petroleum
- ConocoPhillips
- EMGS
- Gaz de France
- Hess
- Kuwait Oil
- PetroBras
- Petro Canada
- PetroChina
- Shell
- StatoilHydro
- Total
- Woodside

- CERN
- DoD, US
- DoE, US
- ENEA
- AWE
- Georgia Tech
- Harvard Medical School
- Japan Atomic Energy Inst.
- MaxPlanck Inst.
- MIT
- SSC, China
- Stanford Medical
- TACC
- U. Tokyo
- Washington U.

- Abott Labs
- AstraZeneca
- Celera
- DuPont
- Eli Lilly
- Johnson & Johnson
- Merck
- National Institutes of Health
- Novartis
- Partners Health Network
- Pharsight
- Pfizer
- Sanger Institute

Other Industries

AT&T Bell Canada DreamWorks Animation SKG GE
IRI Telecom Italia Telefonica Walt Disney Co.

Introduction: CERN requirements & solutions

- Efficient Scheduling of a workload mix with sequential and parallel jobs of different requirements and runtimes
- Platform LSF has been developed for exactly this purpose, providing technology leadership and best performance.
- Several properties and functionalities of Platform LSF are used in conjunction to assure
  - Responsiveness to the user
  - Best possible management of workload
  - Best possible management of resources
- Beyond the mature classical concept, alternatives are discussed
- Combinations of Virtualization with Grid Technology
  - Dynamic inclusion of VMs in the batch cluster
  - Dynamic “personality” change of compute nodes
  - Virtually private Grids within (and beyond) shared resource pools
- The latter topics are related to the talk of Prof. Dr. Sebastien Goasguen.

Introduction: CERN requirements & solutions

- Understanding CERN's workload mix
  - 94% sequential (1-way) jobs
    - typical runtimes of about 1 hour (= between 10 minutes and 10 days)
  - 5% parallel workload, 2-8 tasks/job (2-8 way)
    - Runtimes? (probably similar as sequential case)
  - 1% parallel workload beyond 8 tasks/job (>8 way)

Introduction: CERN requirements & solutions

- Understanding CERN's workload mix
  - Is it fully HPC?
    - Up to now, most sequential jobs were not HPC
    - Optimum host utilization achieved by slight over-commitment: 10 job-slots on 8 core machine,
      - potentially tunable to 12:8
  - In future: change in workload type?
    - MPI jobs with static 1:1 ratio task:job-slot
    - Compute heavy HPC jobs
    - Change from job-slot scheduling to load based scheduling: optimum for many-core nodes

Introduction: CERN requirements & solutions

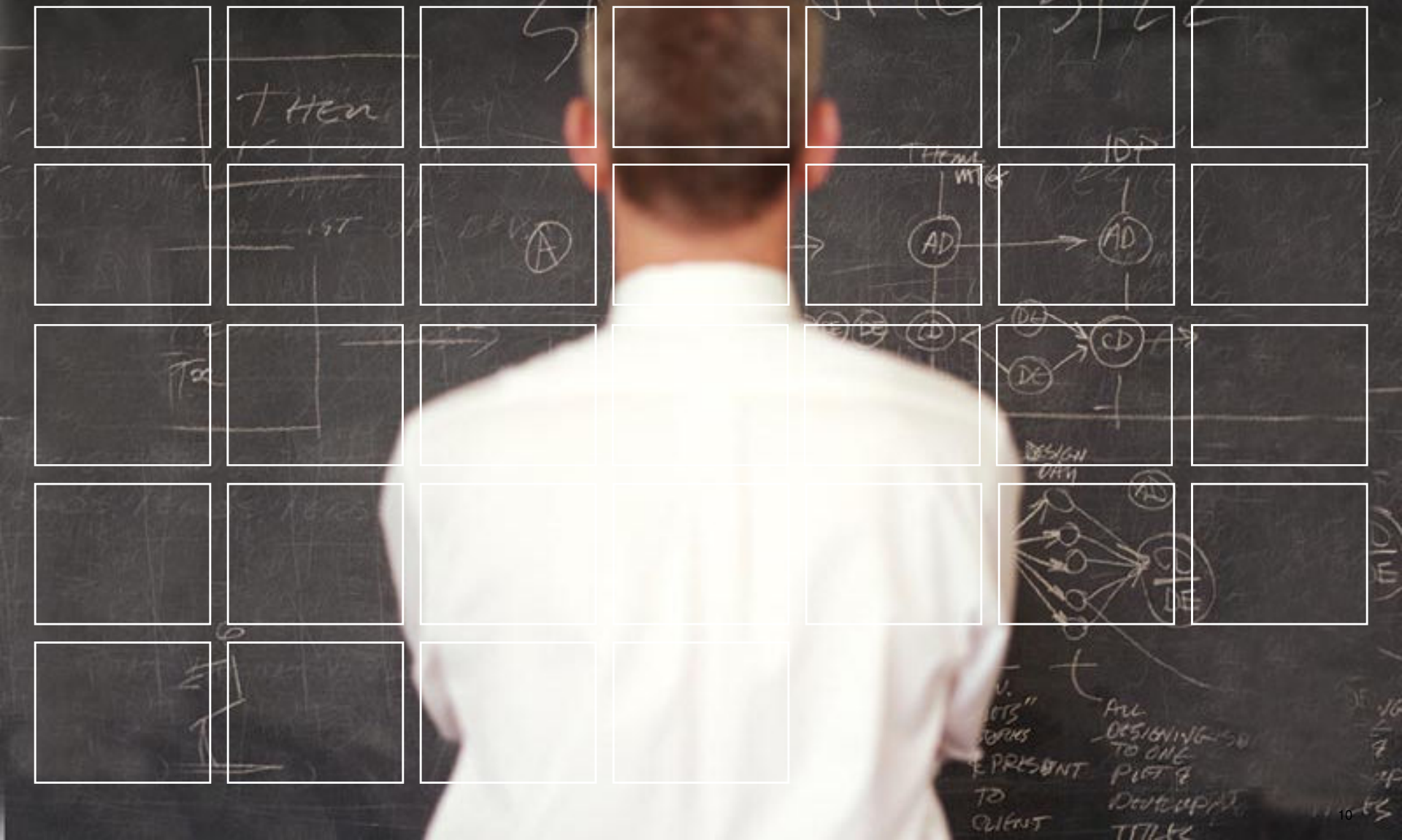
- Further: growing heterogeneity from resource-, application- and usage-pattern-side
- Homogeneous resources are local and transient – entropy works against you!
  - You cannot buy the same HW as 8 weeks ago – the new one is different!
- The number of applications with equal resource requirements is close to zero, compatibility (= the least common denominator) is hard and costly to find.
  - Compromises make many users unhappy
    – and leave admins with the blame ;-)

Introduction: CERN requirements & solutions

- Different user groups develop different workload pattern
  - Need different usage & scheduling policies
- Real world is heterogeneous!
  - Different OS's, versions, patch levels, conflicting configurations, libraries, pre-installed frameworks, incompatible services, services incapable of coexistence on the same host, => examples from the audience?

Platform™

Maximizing the efficiency on multicpu/multicore machines : Managed process binding



Managed process binding on Multi-Core systems

- Nehalem processor is more than 50 percent faster than the previous generation.
- Still jobs using Platform LSF processor binding run up to 12% faster on Nehalem.
- Platform LSF initiated hard processor binding allows you to take advantage of the power of multiple processors, cores and threads.

Processor Affinity

With Platform LSF7update4 processor affinity controls are implemented for:

- Intelligent job placement for single-host, sequential jobs
- Intelligent job placement for single-host, parallel jobs
- Intelligent placement support for multiple-host, parallel jobs
- Five new configurable “packing options”
- Options can apply at the Processor, Core or Thread levels
  - BALANCE
  - PACK
  - ANY
  - USER
  - USER_CPU_LIST

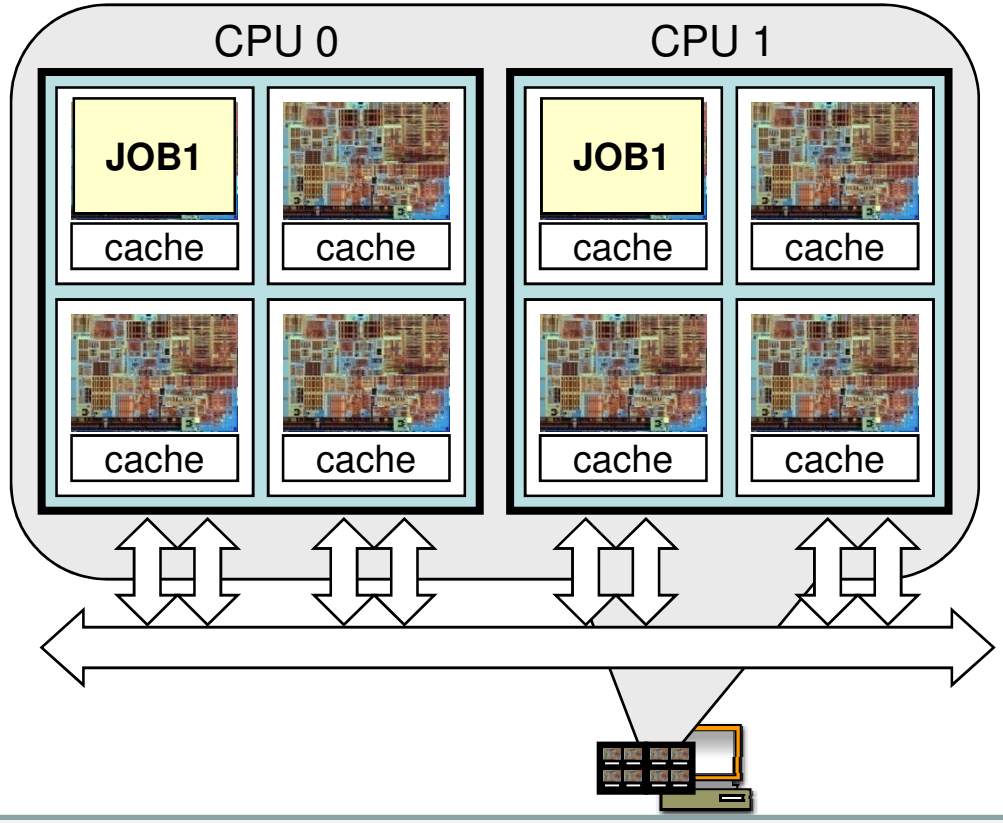
Granular controls over job placement allow cluster users and administrators to take maximum advantage of modern multi-core, multi-threaded processors

Scheduling Enhancements

```
$ bsub -app balanced_job -n 2 -R "span[hosts=1]"
myparjob.sh
```

lsb.applications

```
Begin Application
NAME = balanced_job
BIND_JOB = BALANCE
..
End Application
```



When **LSF_BIND_JOB** is set to **BALANCE**, the scheduler will seek to balance parallel job elements across processors as shown ..

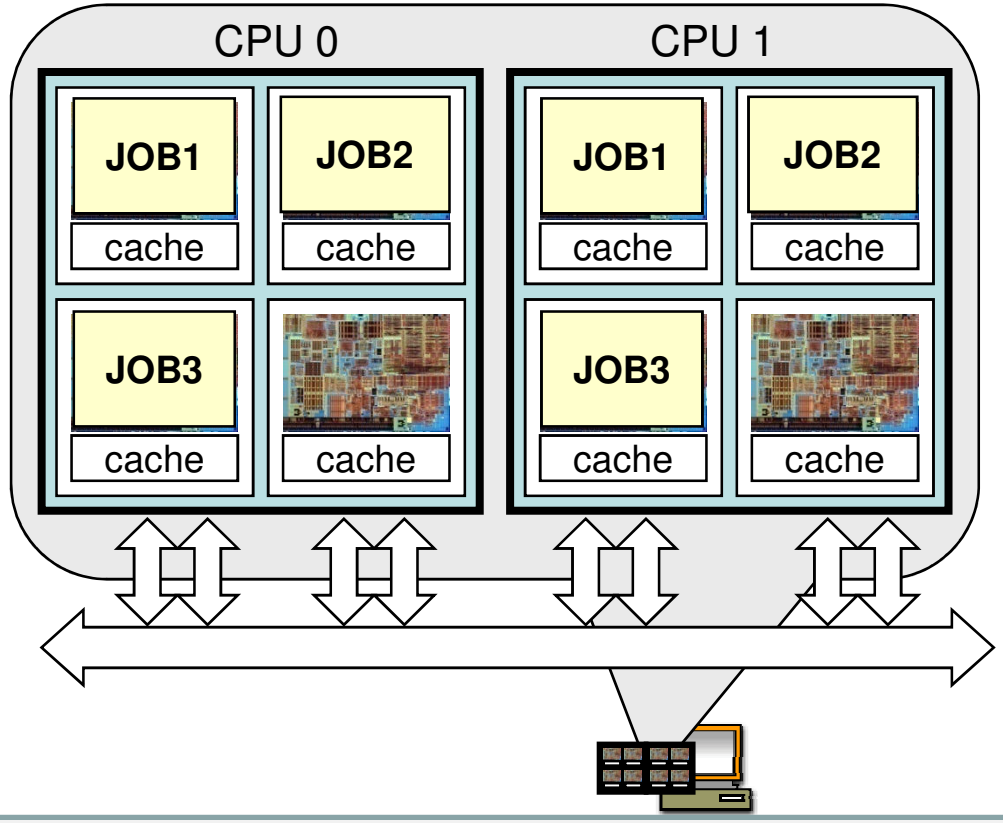
Platform Scheduling Enhancements

Submitting two more jobs with **`-R "span[hosts=1]"`** yields

…

lsb.applications

```
Begin Application
NAME = balanced_job
BIND_JOB = BALANCE
..
End Application
```



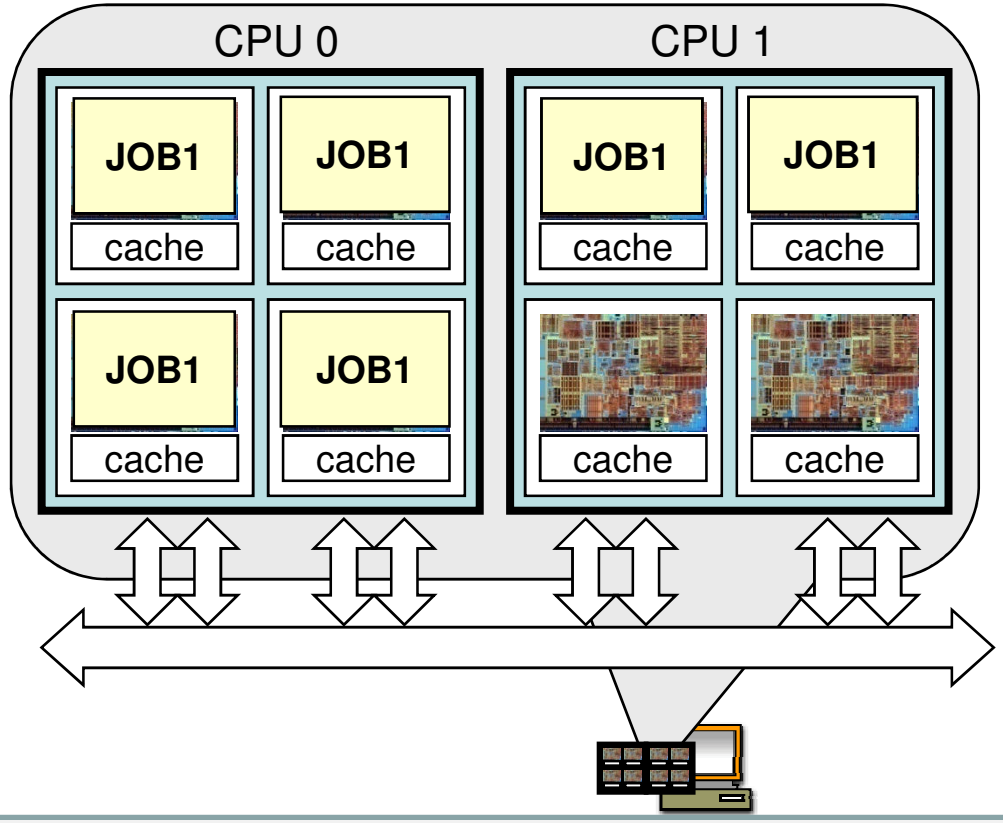
When **LSF_BIND_JOB** is set to **BALANCE**, the scheduler will seek to balance parallel job elements across processors as shown ..

Platform Scheduling Enhancements

```
$ bsub -app packed_job -n 6 -R "span[hosts=1]"
myparjob.sh
```

lsb.applications

```
Begin Application
NAME = packed_job
BIND_JOB = PACK
..
End Application
```



When **LSF_BIND_JOB** is set to **PACK**, the scheduler will seek to use the smallest number of processors possible to run a parallel job ..

Configuration

- LSF_BIND_JOB
  - Five processor binding options: BALANCE, PACK, ANY, USER, USER_CPU_LIST
  - **in lsf.conf**
    LSF_BIND_JOB =**NONE|BALANCE|PACK|ANY|USER|USER_CPU_LIST**
  - **lsb.applications**
    BIND_JOB = **NONE|BALANCE|PACK|ANY|USER|USER_CPU_LIST**
    The application profile overrides lsf.conf configuration if both configured.

- Environment variables specifying the binding request at submission
  - Allow end user specifying binding request through submission environment variables: LSB_USER_BIND_JOB, LSB_USER_BIND_CPU_LIST.

Binding processes

- Sequential job
  - LSF binds processes
- Multi-process single host job
  - LSF binds processes
- Multi-process multi host (distributed) job
  - LSF does not bind processes at all if a job is across multiple hosts
  - Instead, LSF sets the environment variables ($LSB_BIND_JOB and $LSB_BIND_CPU_LIST) on the 1st execution host
  - It's up to the application whether or not to bind processes using those environment variables

Platform™

How to guarantee that parallel jobs can run in mixed sequential/parallel environment

Scheduling = competition !

- Sequential jobs will always win because:
  - Sequential jobs can run as soon as there is a slot available
  - Parallel jobs must acquire enough resources (slots) for being able to start.

- In a busy cluster (pending jobs) with a majority of sequential jobs, if parallel jobs just keep waiting for slots, chances that x slots are freed up at the same time are almost zero, so sequential jobs will «steal» all available slots.

Solution: slot reservation

- If LSF is instructed to do so, slots which become free will be « reserved » for pending parallel jobs.
- Slots will be reserved during a certain amount of time; if the job still can’t start after this delay, reservation is cancelled and the reservation process restarts from the beginning.

Backfilling

- LSF predicts the starting time of a parallel job using the runtime limit of currently running jobs.
- Small jobs can use the “backfill window” if their runtime limit guarantees that they will finish before the predicted starting time.
- In this context it becomes important to specify runlimits !

Displaying the backfill window

```
bslots
SLOTS           RUNTIME
2               16 minutes 14 seconds
5               11 minutes 31 seconds

bslots -n 4
SLOTS           RUNTIME
5               11 minutes 17 seconds

bslots -W 14
SLOTS           RUNTIME
2               15 minutes 47 seconds
```

Generalized resource reservation

- Example: pending jobs in order of priority:
  - job1 needs resource1, resource2 and resource3
  - job2 needs resource1 and resource2
  - job3 needs resource1
  - job4 needs resource1
  - job5 needs resource2
  - job6 needs resource2

If only resource1 is available, LSF will allow job3 and job4 to run; similarly, if only resource2 is available job5 and job6 will run – this is good for throughput. However, if resource3 is scarce, job1 will likely never run as other jobs will be using resource1 and resource2.

Generalized resource reservation

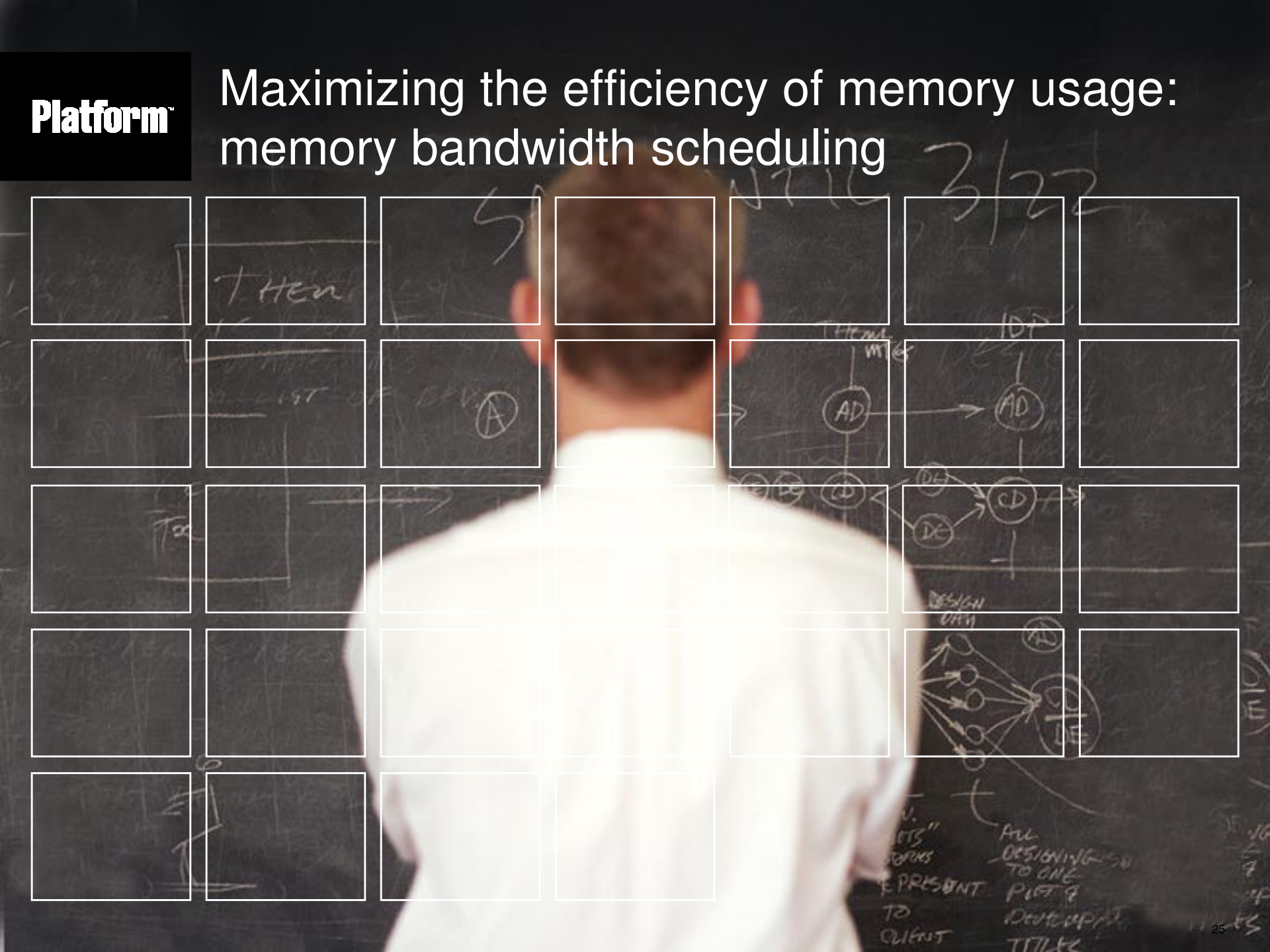
```
Begin ReservationUsage
RESOURCE                METHOD                RESERVE
mem                     PER_HOST              Y
tmp                     PER_HOST              Y
lic1                    PER_SLOT              Y
lic2                    PER_JOB               N
End ReservationUsage
```

Resources: memory, temp space, lic1 will be reserved along with slots.

Reserved resources can be backfilled.

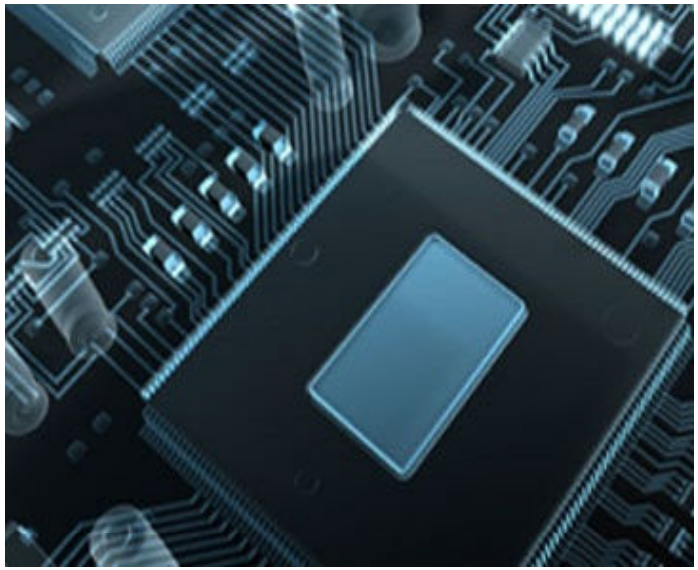
Platform™

Maximizing the efficiency of memory usage: memory bandwidth scheduling



Why memory bandwidth aware scheduling ?

- Quad-cores are currently available
- As the number of cores on a processor increases, clock speed will decrease



- Serial applications will have to be parallelized to achieve the same application performance
- CPU alone is not the only performance factor; memory bandwidth is a critical resource.

Platform Why memory bandwidth aware scheduling ?

- Achieving adequate performance is relatively easy to obtain with full control of the host, but dispatching heterogeneous jobs may lead to major inefficiencies.

- We want to be able to schedule jobs based on their memory bandwidth.

➔ Bandwidth aware scheduling

How to: memory bandwidth scheduling

- An LSF “bw” resource is created and assigned to each node.
- The value of the resource can be configured on a host-by-host basis.

Platform LSF* can automatically manage the demand vs. supply of memory bandwidth

- The resource is shared among the jobs running on the server.

Steps:

1. Profile the hosts

   Can be done with the STREAM standard benchmark

2. Profile the applications

   Run the STREAM benchmark along with the application and record the difference.

3. Configure LSF

How to: memory bandwidth scheduling

- Create the Platform LSF* memory bandwidth resource

```
# Edit the lsf.shared file
Begin Resource
RESOURCENAME TYPE INTERVAL INCREASING DESCRIPTION
bw Numeric () N (available bandwidth)
End Resource
```

Simple Configuration

How to: memory bandwidth scheduling

Initialize the Platform LSF* memory bandwidth resource on each server

```
# Edit the lsf.cluster.clustername file to
HOSTNAME   model     type    server RESOURCES
hosta      !          !       1      (bw=6180)
hostb      !          !       1      (bw=12000)
hostc      !          !       1      (bw=6180)
```

Simple to Use with Amazing Results

How to: memory bandwidth scheduling

- Reconfigure the cluster

```
lsadmin reconfig
badmin reconfig
```

- Submit a job

```
bsub -R "rusage[bw=800]" myjob
```

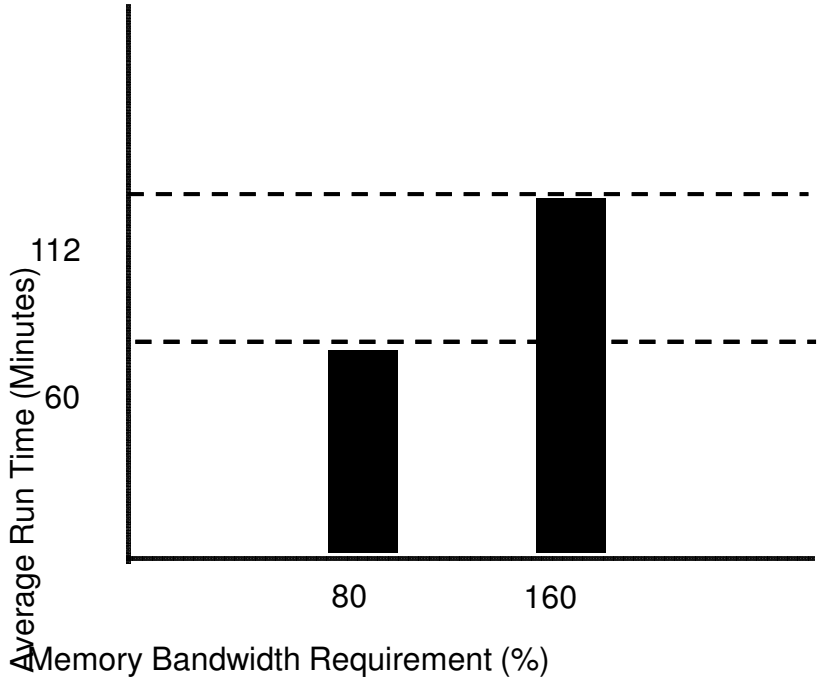
Minimize turn-around time for high priority jobs

Or Maximize cluster throughput

Platform Results: memory bandwidth scheduling

Tested on HP DL140 ProLiant* servers with dual Intel® Xeon® 5140 Processors

- Ran two sets of four jobs:
  - Two jobs at a time
  - Then four jobs at a time
- Average run time shorter when bandwidth (bw) undersubscribed
- Turn-around time for the set was shorter when bandwidth was oversubscribed



→ Minimize turn-around time for high priority jobs

Or → Maximize cluster throughput

Platform

Tuning: memory bandwidth scheduling

- To minimize turn-around time
  - The resource should be set so that the application is not limited by memory BW
  - Example: bw = actual bw x bw utilization factor
    bw = 6180 x .9 ~ 5500 MB/sec

Enable More Efficient Use Of Application Licenses

Platform

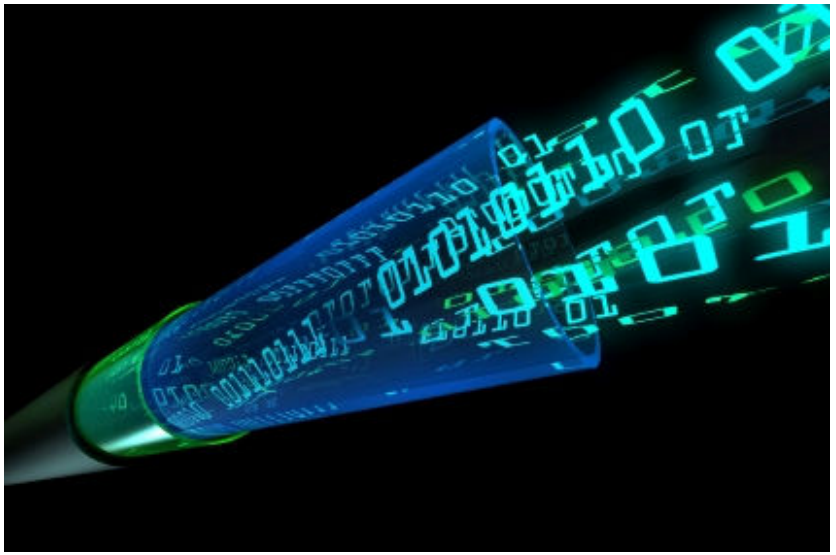
Tuning: memory bandwidth scheduling

- To maximize throughput
  - Slightly over-subscribing BW results in a shorter turnaround time for a set of jobs
  - Example:
    bw = Actual bw x bw utilization factor
    bw = 6180 x 1.1 ~ 6800 MB/sec
- Individual jobs will take longer to complete, but throughput is maximized

Maximize cluster throughput

Limitations: memory bandwidth scheduling

- Its hard to know how much memory bandwidth a job will require
- There is no way to force a job to live within its allocation



- Poor performance if user under-estimates the bw resource requirements

Memory Bandwidth is difficult to determine

Platform Resources: memory bandwidth scheduling

Need to learn how to ascertain how much memory bandwidth a job is using.

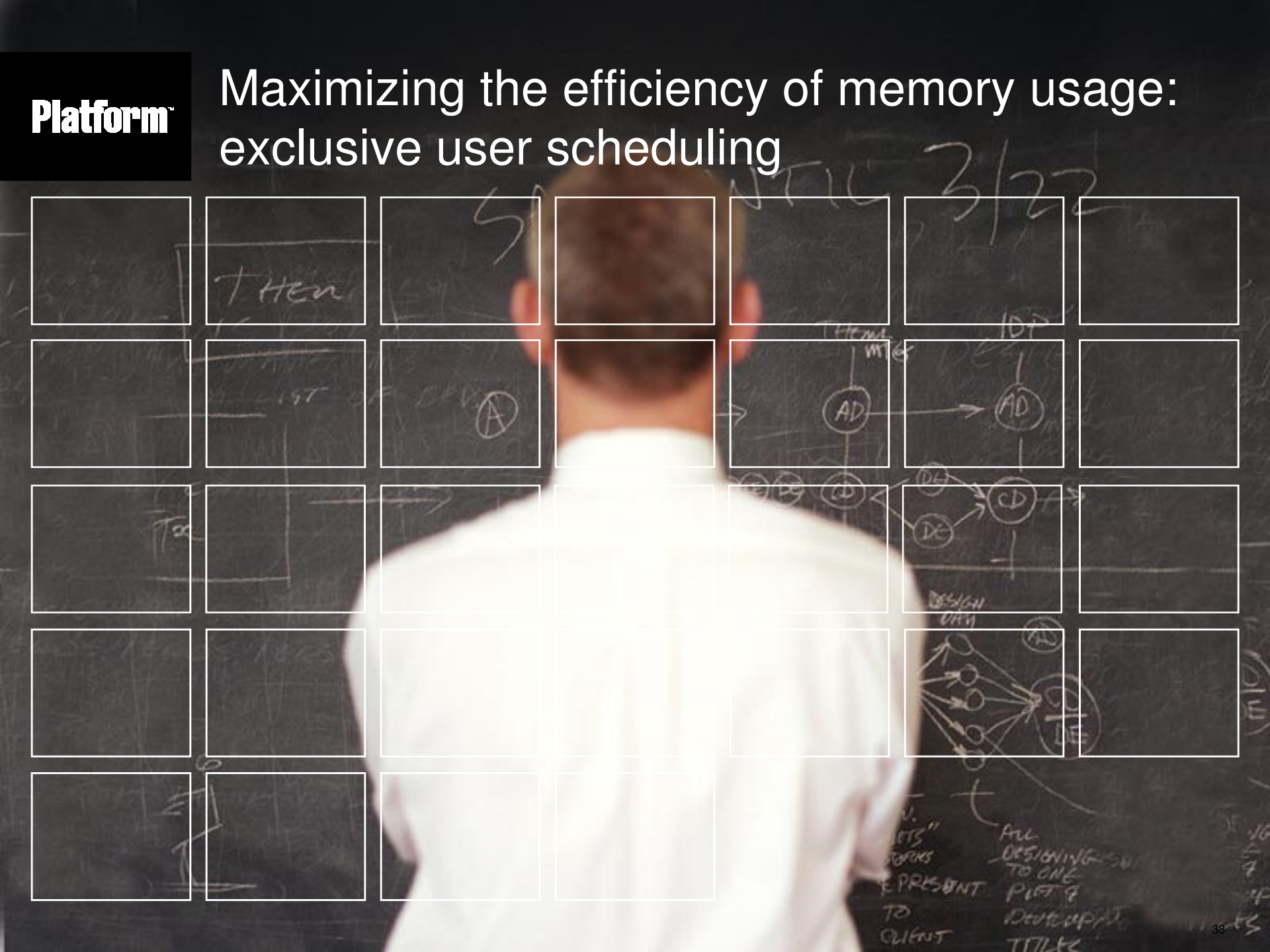
- Download HP – Platform white paper “Scheduling to Overcome the Memory Bandwidth Bottleneck”

http://www.hp.com/techservers/hpccn/hpccollaboration/ADCatalyst/downloads/Platformmulticore_scheduling_R11FINAL.pdf

http://www.hp.com/techservers/hpccn/hpccollaboration/ADCatalyst/multi-core/partners/Platformcomputing.html

Platform™

Maximizing the efficiency of memory usage: exclusive user scheduling



Maximizing the efficiency of memory usage

- Cloud style on demand allocation of hosts for exclusive use by a specific user
  - exclusive user scheduling
  - Stay alone on your host!
- Motivation: a Linux bug-by-design
  - http://www.csamuel.org/2007/11/30/linux-ulimit-memory-enforcement-oddity
  - Since Linux does not control mem usage, processes of different users compete for the local memory.
- Solution 1: fix Linux (out of scope today ;-)
- Solution 2: use LSF mem-limit enforcement
- Solution 3: limit the damage: isolate the users – let their own processes compete for the local memory.

User exclusive scheduling

- Solution 3: limit the damage: isolate the users – let their own processes compete for the local memory.
  - != bsub –x … → that would block a host for one job
  - We want several jobs of one user to be run exclusively on a host
- Exploit the LSF arbitrary semantics infrastructure
  - An ELIM per host reads out the users and populates a new “LSF resource”
  - An ESUB modifies the job submission to add the job requirement –R “act_user==<me> || act_user==NONE“ to express I need a host with <me> as user or NONE

User exclusive scheduling: config

- Here is the config:

lsf.shared:
Begin Resource
RESOURCENAME TYPE INTERVAL INCREASING
DESCRIPTION
act_user String 10 () (Actual user on this node)
End Resource

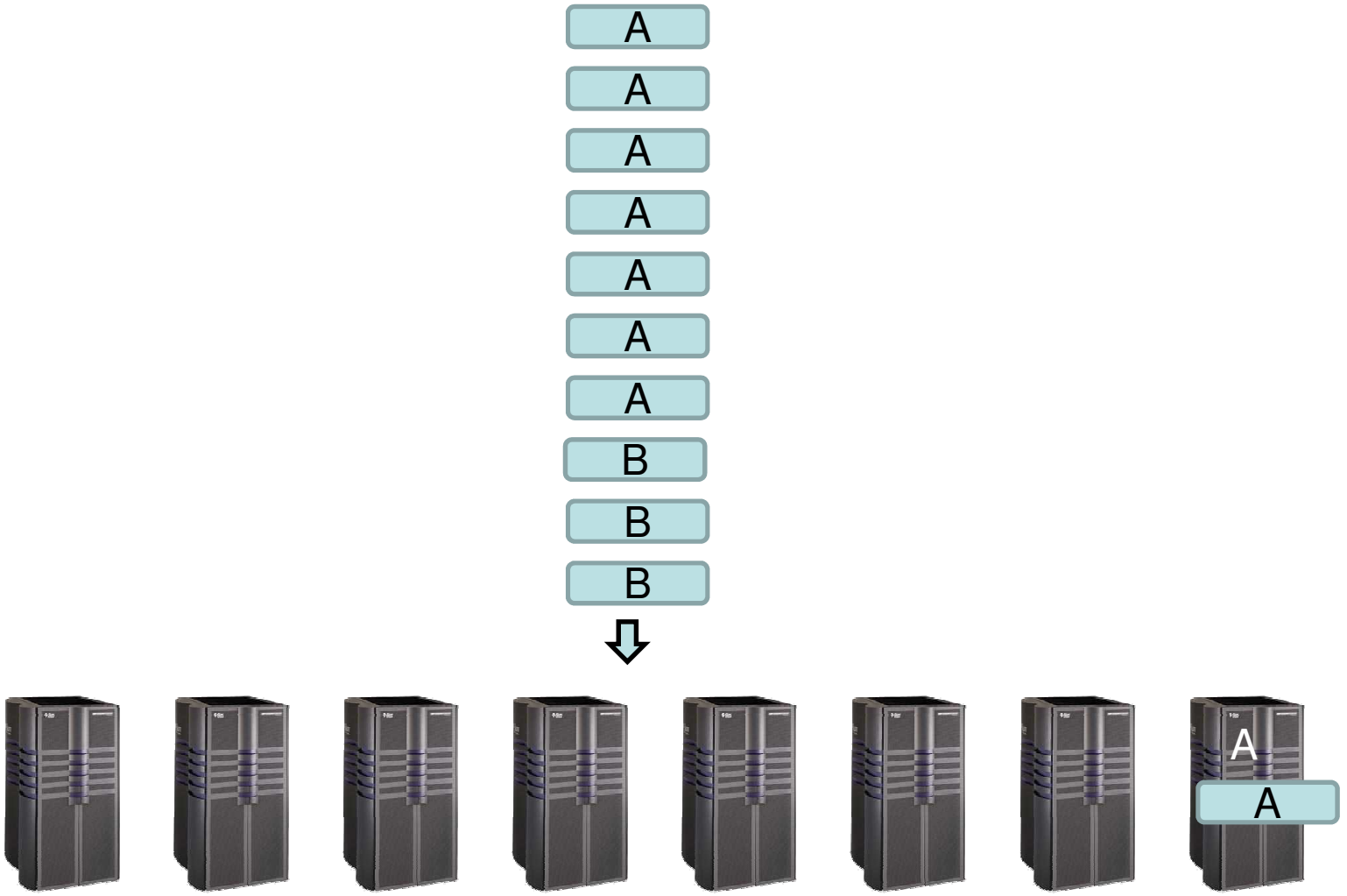
lsf.cluster.<clustername>:
Begin ResourceMap
RESOURCENAME LOCATION
act_user [default]
End ResourceMap

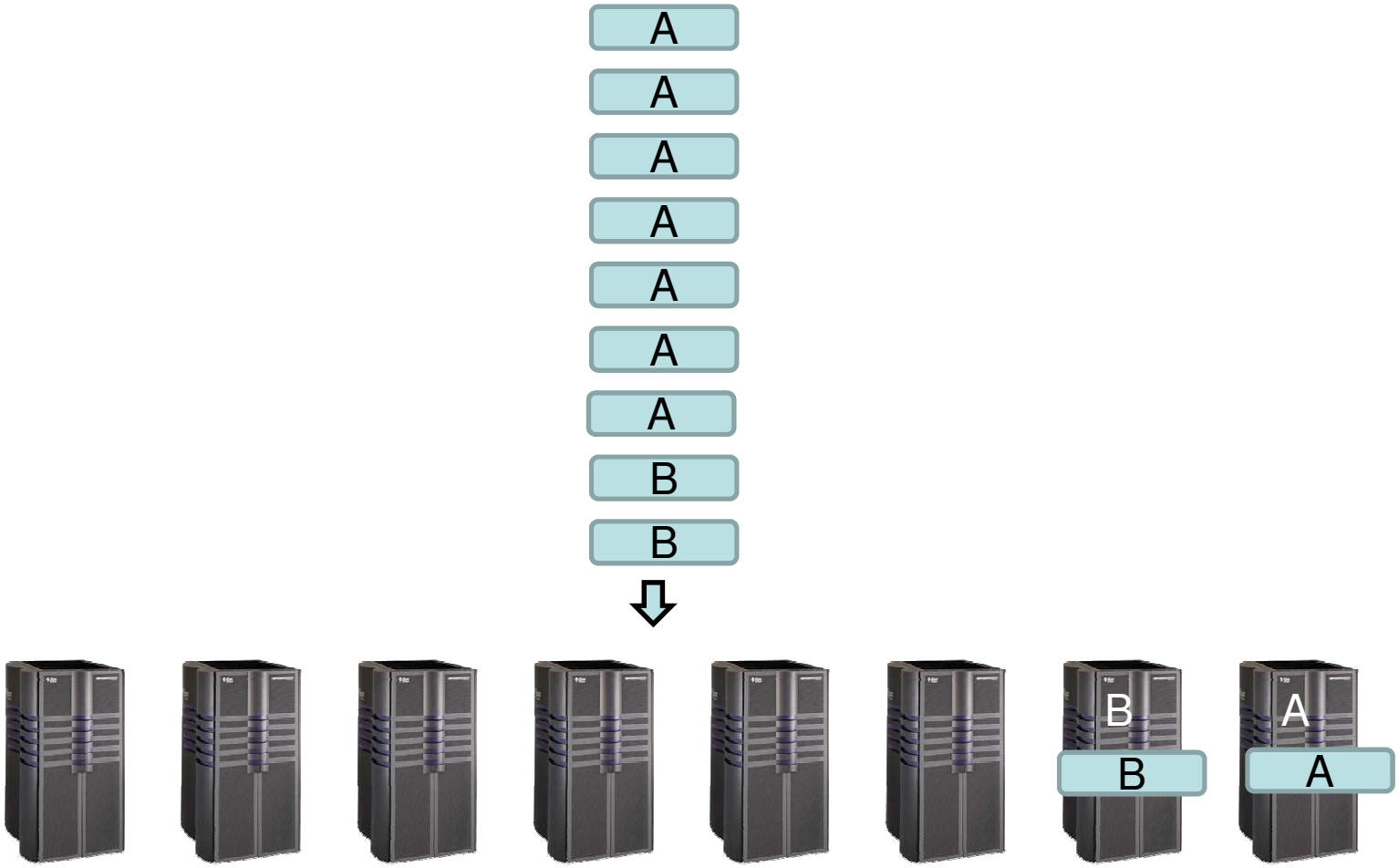
User exclusive scheduling: ESUB & ELIM

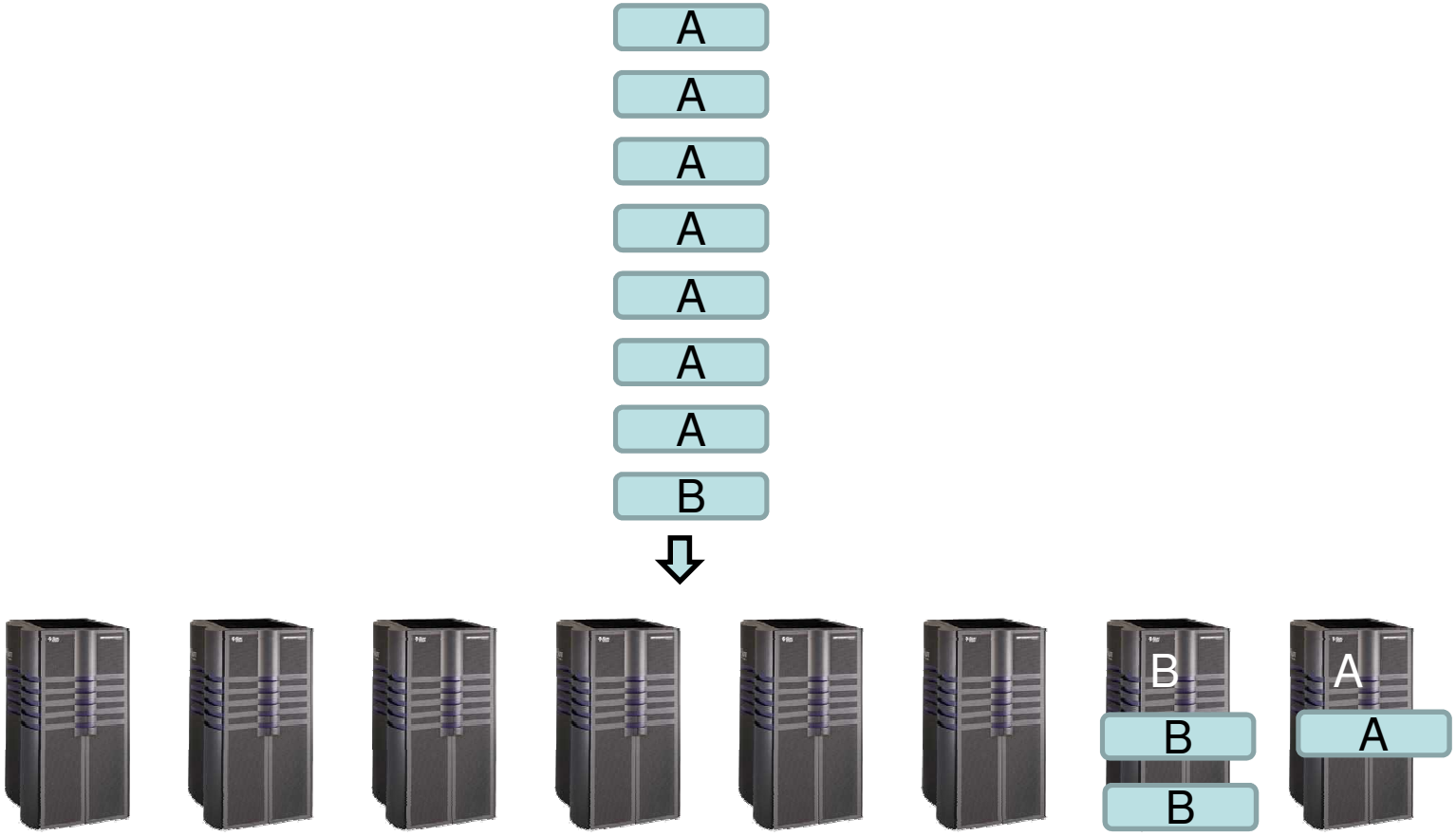
```
esub:
#!/bin/sh
if test "$LSB_SUB_PARM_FILE" != ""
then
   . $LSB_SUB_PARM_FILE
   eval  echo 'LSB_SUB_RES_REQ="\"select[(act_user==${USER})||(act_user==NONE)]\""' >
$LSB_SUB_MODIFY_FILE
fi

elim:
#!/bin/sh
EXCEPT_LIST="(68|dbus|gdm|root|rpc|USER|xfs|lsfadmin)"
while true
do
   ACT_USER=`ps auxww | awk '{print $1}' | sort | uniq | egrep -v ${EXCEPT_LIST} | head -1`
   if test "${ACT_USER}" = ""
   then
      ACT_USER="NONE"
   fi
   echo "1 act_user ${ACT_USER}"
   sleep 10
done
```

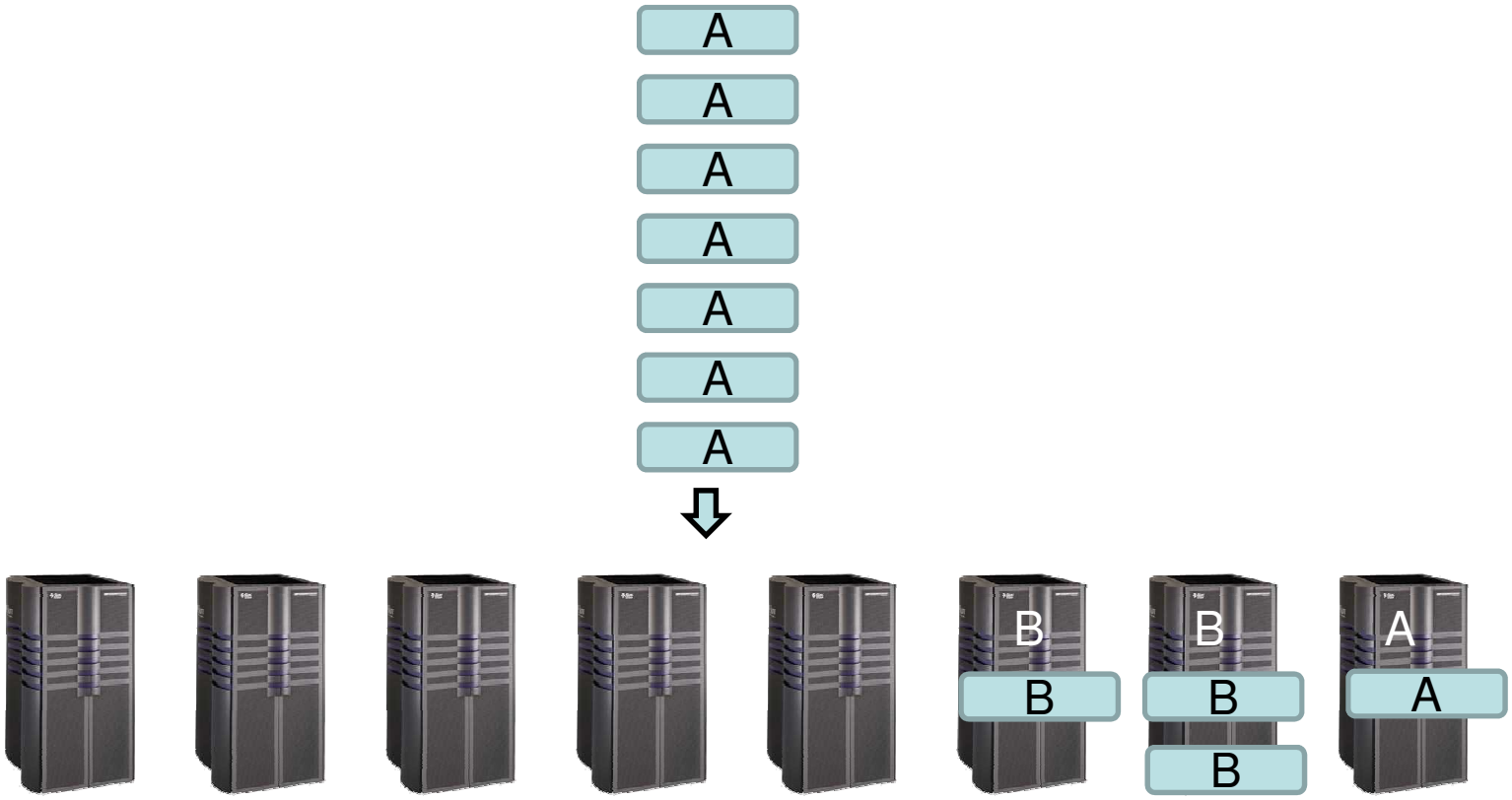
Possible extension: dynamic partitioning

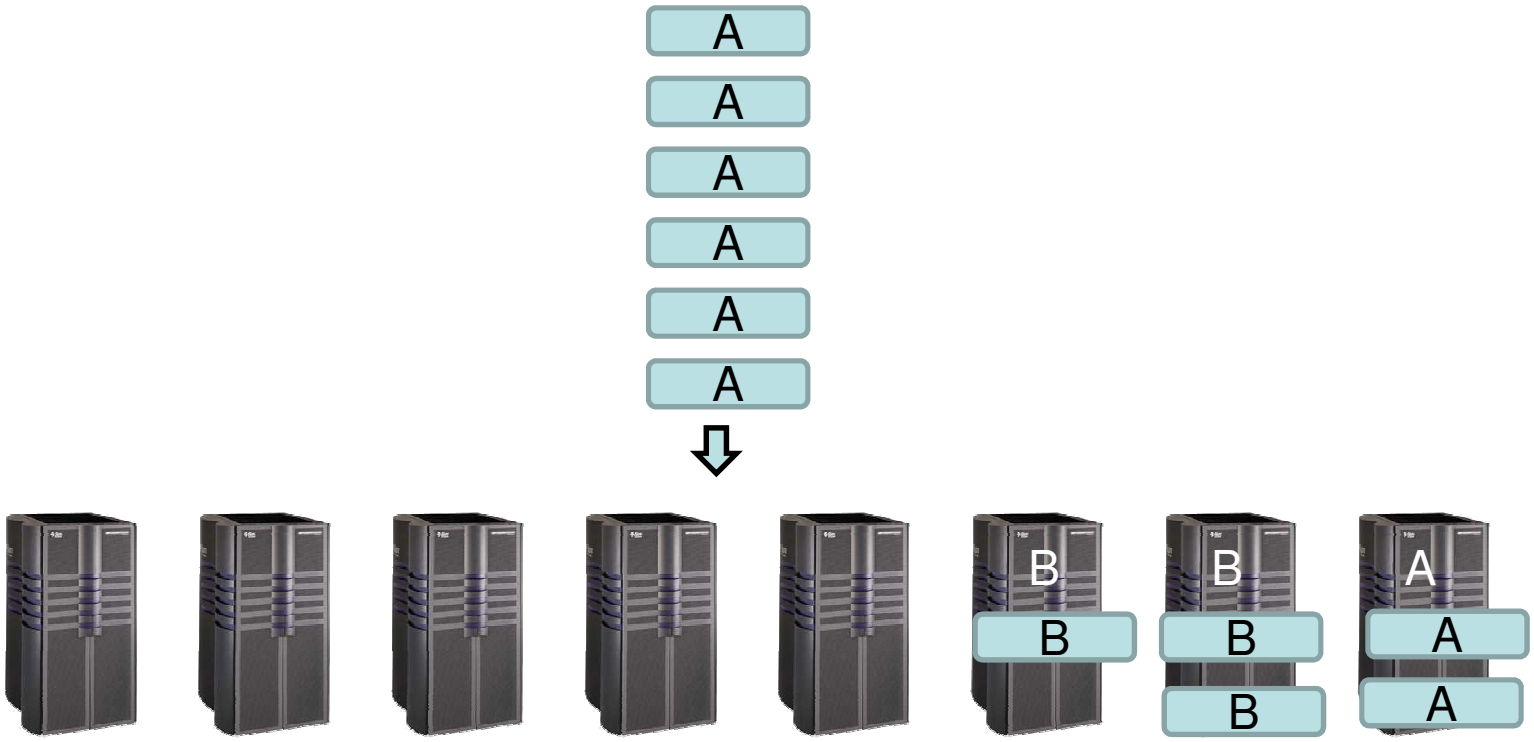
- « User-exclusive scheduling » is a special case of a more general concept: « dynamic partitioning ».
- The previous technique temporarily assigns a host to a specific usage by setting an attribute, triggered by the presence of specific jobs.
- This technique could be extended and take in account any kind of attribute (ex. packed vs balanced hosts, or undersubscribed vs oversbuscribed hosts). Hosts are grouped for similar usage in a totally dynamic way, which is constantly adapting itself to the changing workload.

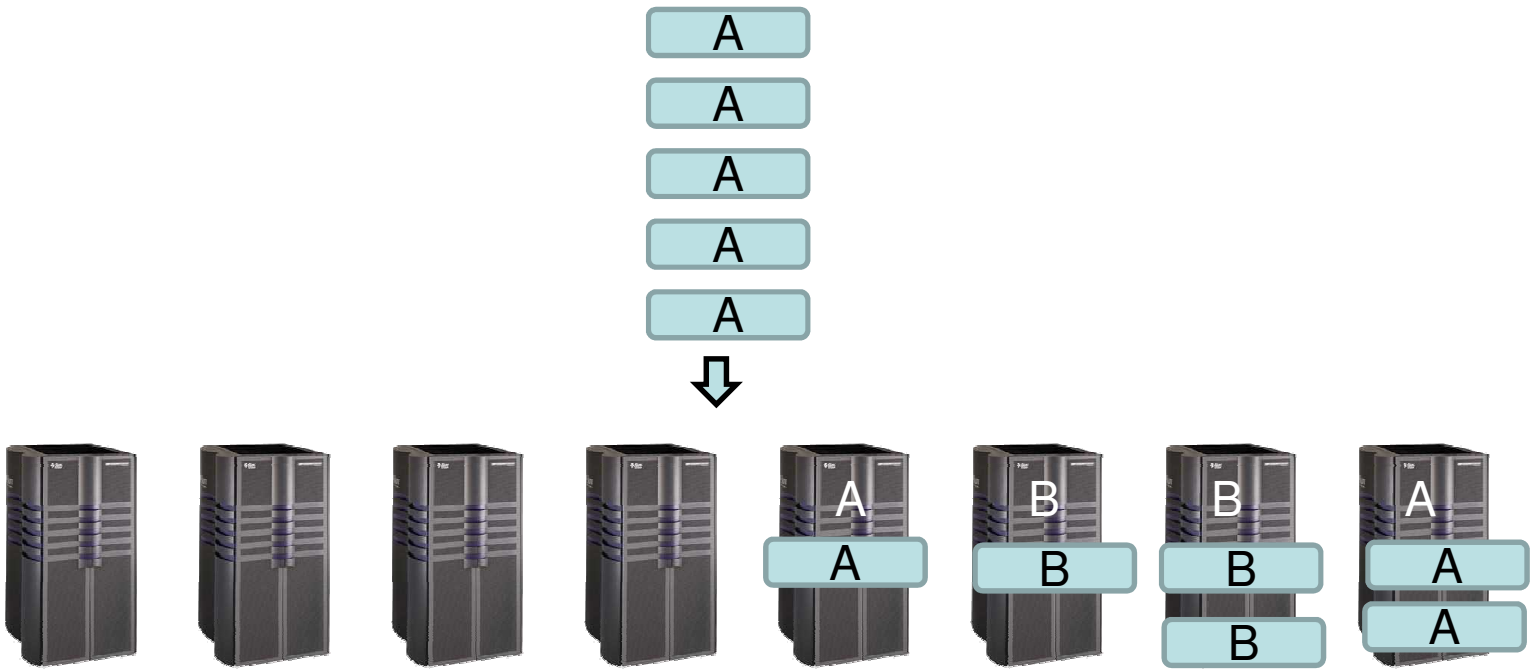


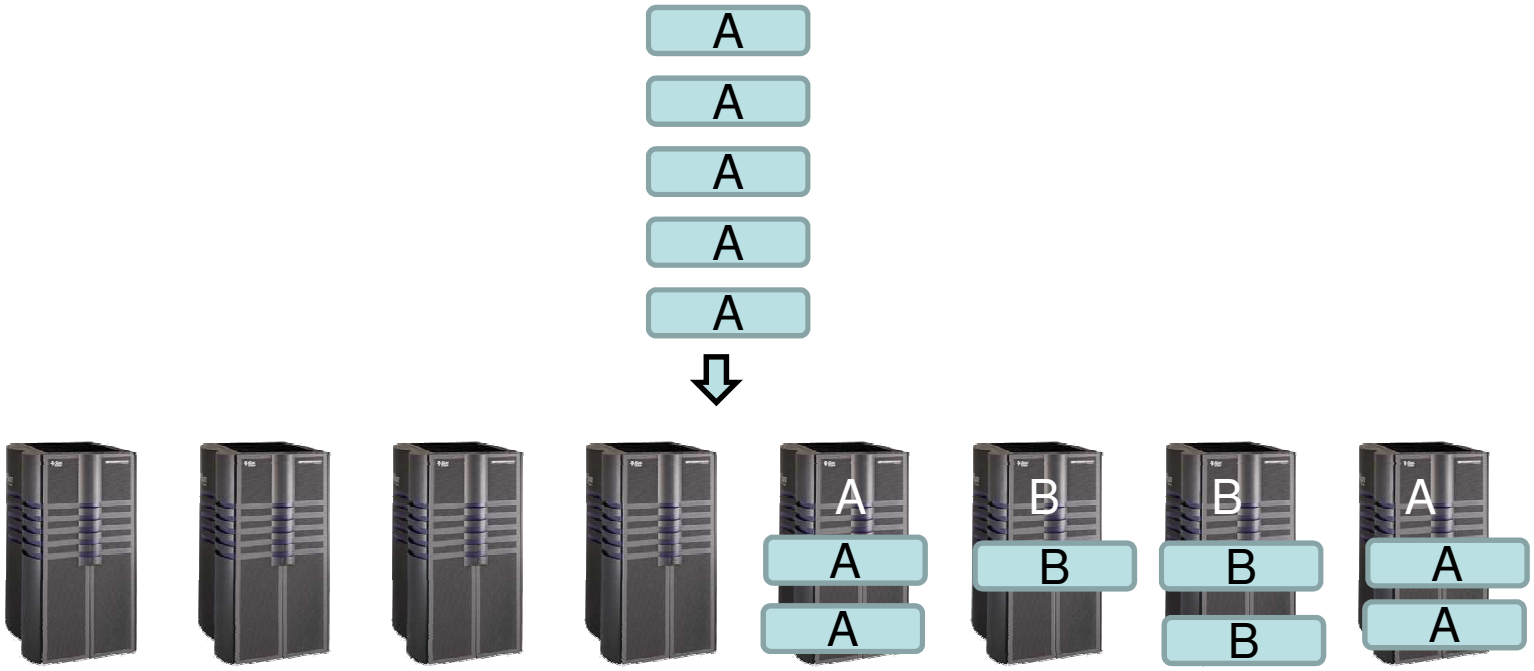


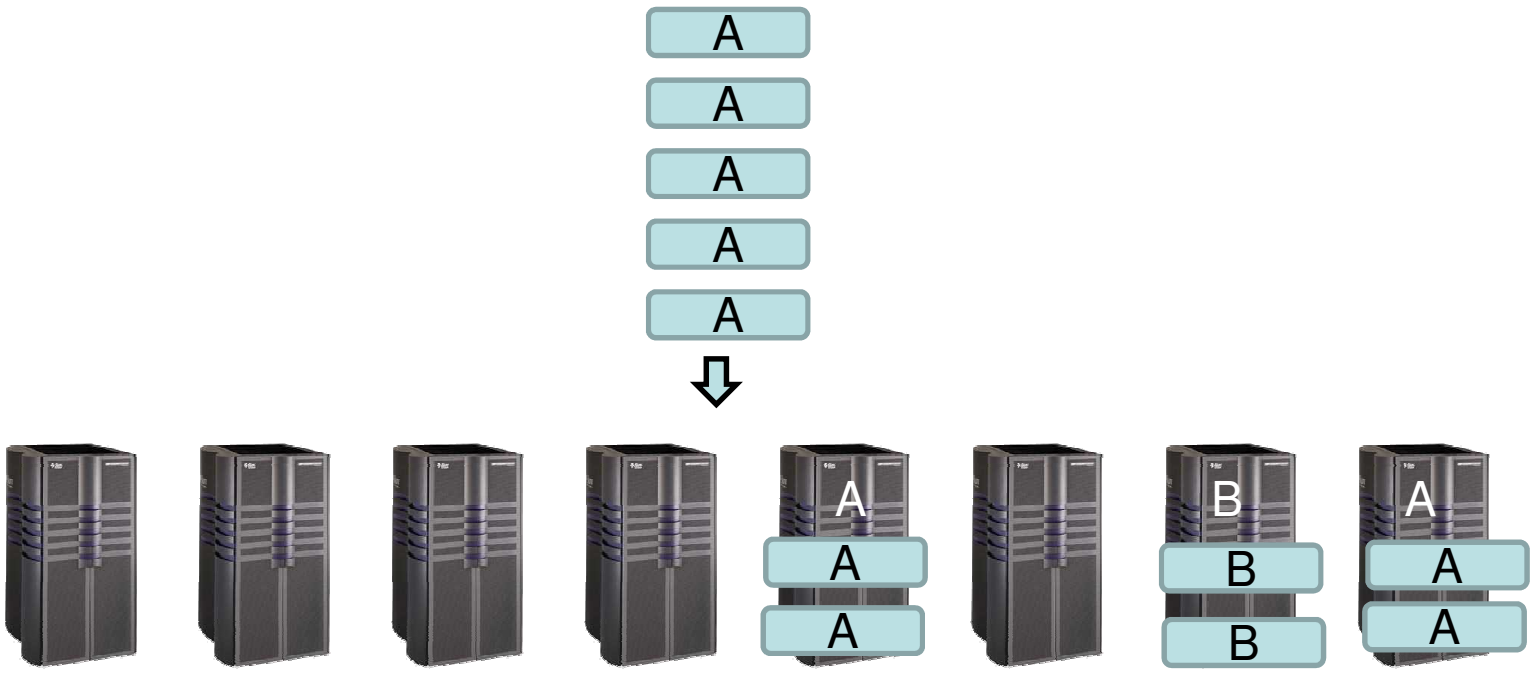


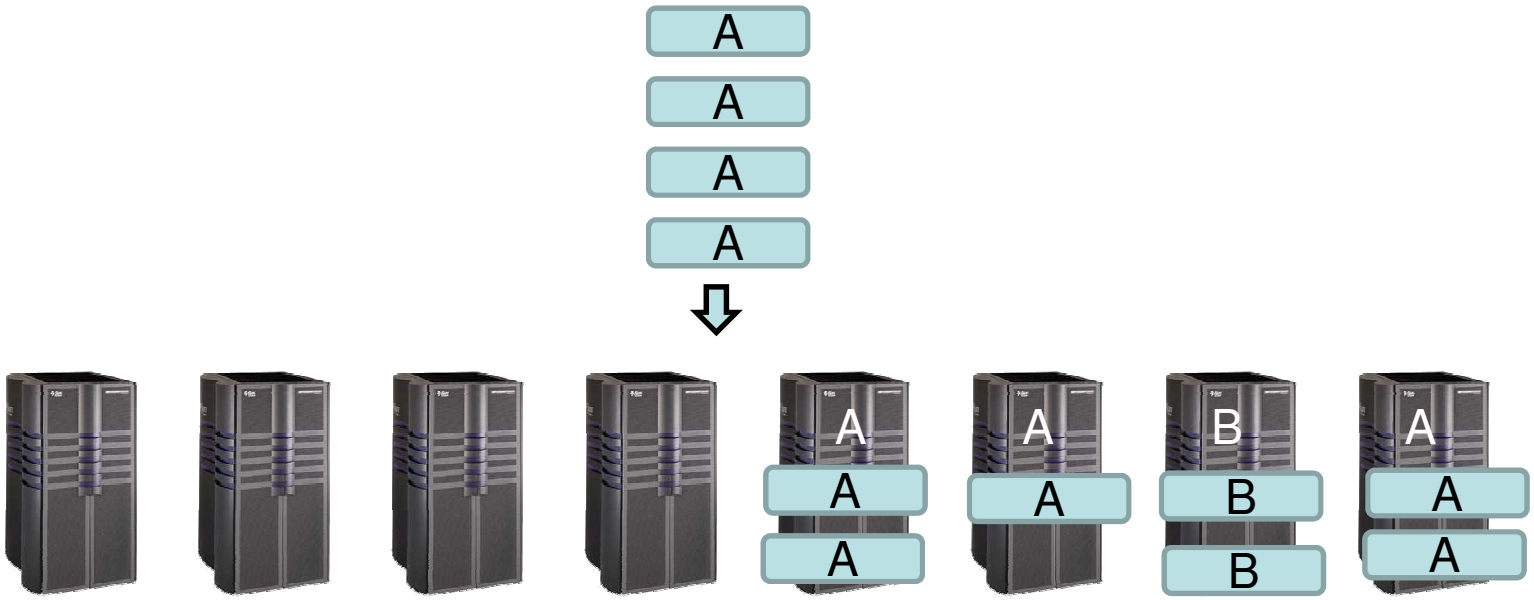








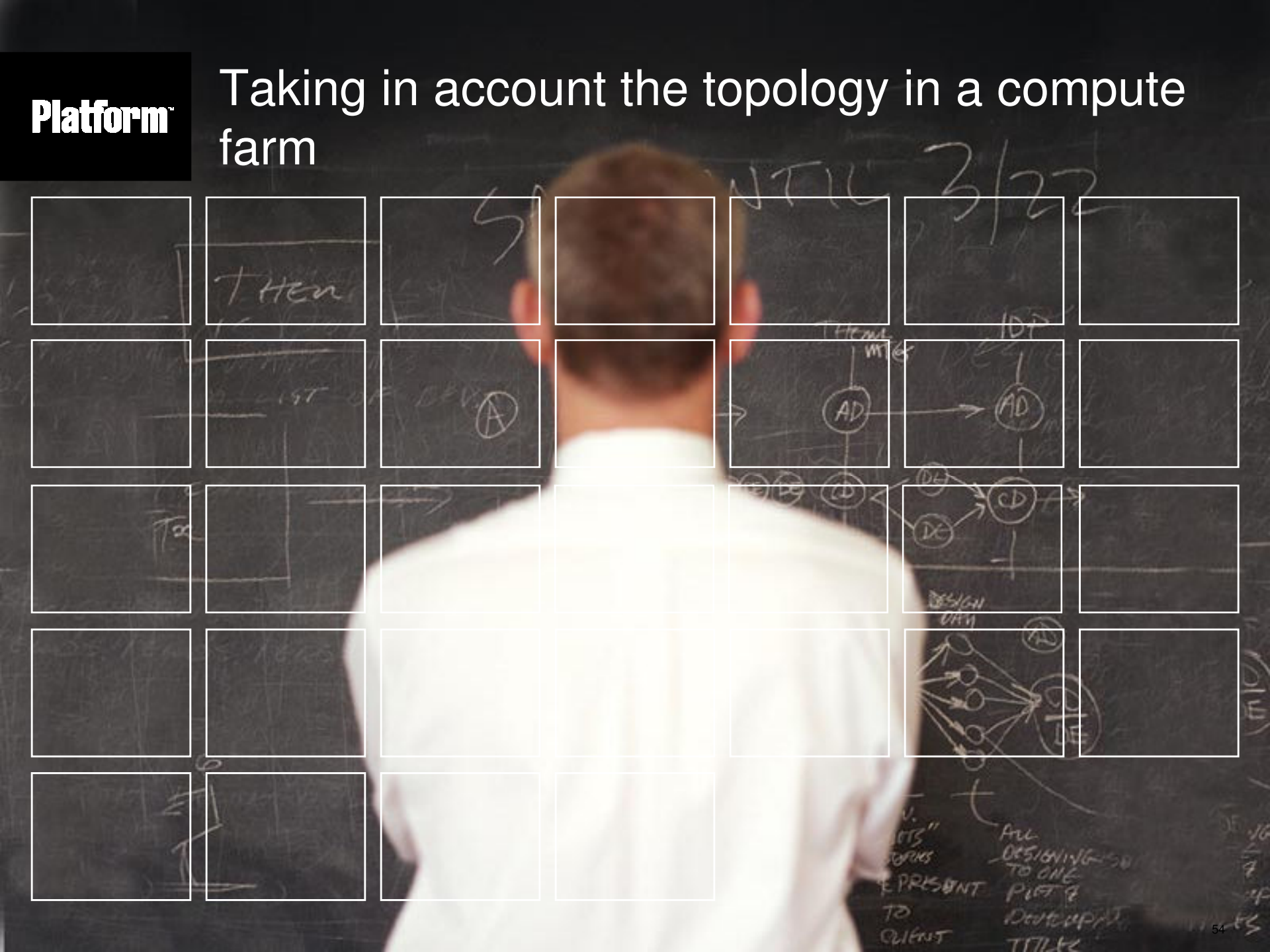




```
Begin Limit
NAME = Undersubscribed_hosts
 #mandatory
PER_USERS = all ~exp1_group     #consumer
Undersubscribed=10          #resource
End Limit
```

Platform™

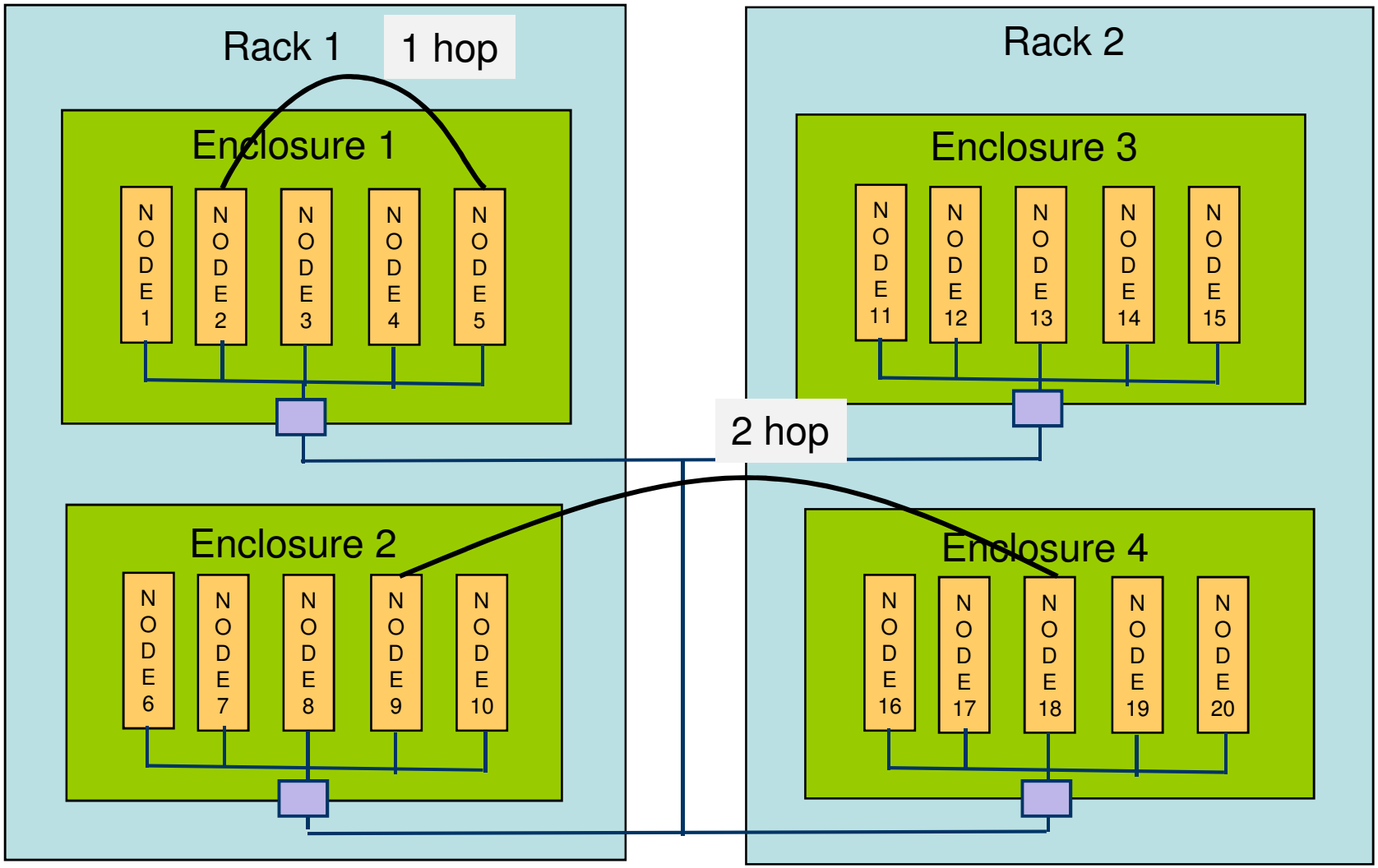
Taking in account the topology in a compute farm



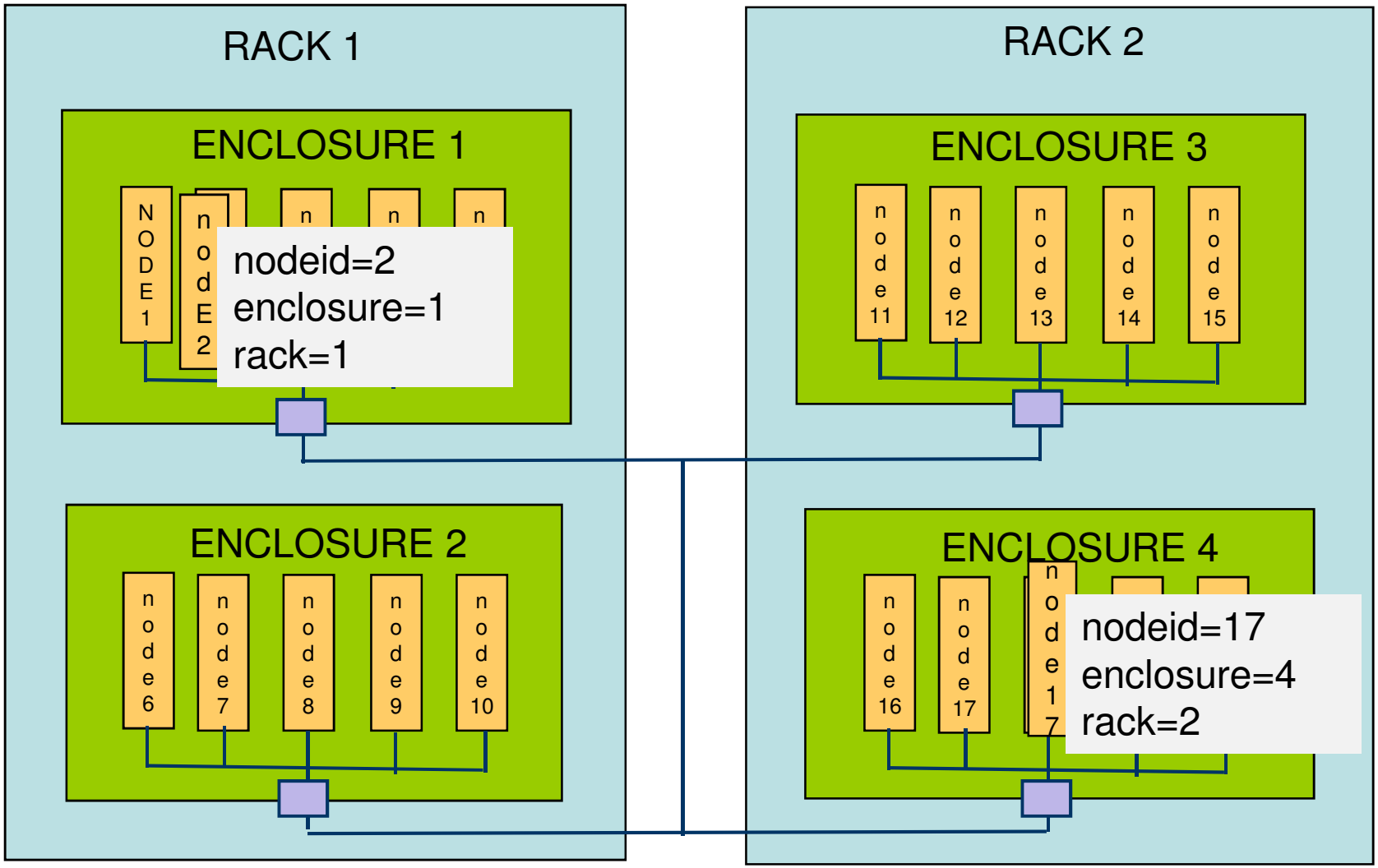
Background

- Many HPC clusters/machines are composed of multiple node groups (e.g. blade server enclosures).
  - each node group consists of multiple nodes (e.g., individual server blades).
- Nodes are connected by high-speed interconnect.
  - within a group, nodes communicate through a local switch.
  - nodes in different groups normally communicate through multiple switches.
- I/O intensive parallel applications: in general, running in few node groups results in better performance.
- Sites want LSF to consider network architecture when scheduling parallel jobs in such environment.

Enclosure-based HPC machine



Host characterisation through static resources



Controlling the allocation for parallel jobs – basic features

- At the host level:
  - *-R "span[hosts=1]"* (force the job to run on a single host)
  - *-R "span[ptile=x]"* (force the job to use x slots per host)

- Taking in account topolgy:
  - *-R "same[]"*, force job run within a single group of nodes that have the same value for a resource.
  - *-R "order[nodeID]"*, place job according to node ID sequence, where nodes' IDs are based on group.

Examples

Keeping jobs within the same unit

bsub –n 64 –R “same[enclosure]”

bsub –n 64 –R “same[rack]”

Keeping allocation as contiguuous as possible

bsub –n 64 –R “order[nodeid]”

Keeping hosts available for big parallel jobs

bsub –n 64 –R “order[nodeid]”

bsub –R “order[-nodeid]”

Going further

Limitations:

- With *same[ ]* the job will not start if 64 slots available but not in the same enclosure, while the user maybe would have accepted sub-optimal performance
- *same[ ]* is not applicable for jobs bigger than one enclosure/rack/…
- *order [ ]* will keep allocations as contiguous as possible but with no guaranty.

Solution ➔ Compute units

Compute unit

```
Begin ComputeUnit
NAME   MEMBER          TYPE
encl1     (host[1-16]) enclosure
encl2     (host[17-32]) enclosure
encl3     (host[33-48]) enclosure
encl4     (host[49-64]) enclosure
rack1     (encl1 encl2) rack
rack2     (encl2 encl3) rack
End ComputeUnit
```

Compute Unit – Resource Requirement

```
bsub -R
"cu[type=:pref=:maxcus=:usablecuslots=:balance:excl]"
```

- *type=<cu_type>:* CU granularity.
- *pref=maxavail|minavail|config* : CU preference.
- *maxcus=<number>* : bound the number of CUs.
- *usablecuslots=<num>* : avoid CUs with less than threshold of slots usable by the job.
- *balance:* balance tasks across CUs.
- *excl:* use CU exclusively.

CU granularity

```
bsub -R
"cu[type=:pref=:maxcus=:usablecuslots=:balance:excl]"
```

– Describes the granularity at which the CU resource requirement is to be applied.

```
bsub -R "cu[type=enclosure]"
```

CU preference

```
bsub -R
"cu[type=:pref=:maxcus=:usablecuslots=:balance:excl]"
```

pref=maxavail|minavail|config

- Determine the order of CUs: CU with higher preference value will be considered first during scheduling.
  - *maxavail*: Prefer CUs with more free slots to those with fewer free slots.
  - *minavail*: Prefer CUs with fewer free slots to those with more free slots.
  - *config*: Preference based on the order that CUs appear in the *ComputeUnit* section.

```
bsub -R "cu[type=enclosure:pref=minavail]"
```

Maximum number of CUs

```
bsub -R
"cu[type=:pref=:maxcus=:usablecuslots=:balance:excl]"
```

maxcus= x

- restrict the maximum number of CUs allocated for the job.
  - prevents job's allocation from spreading across too many CUs, hurting job performance.
- typical use cases
  - *-R "cu[maxcus=1]":* restrict the job to run within a single CU --- similar to *–R "same[]"* functionality
  - *-n x -R "cu[maxcus=y]":* restrict the job to run within *y* CUs.

Minimal usable slots per CU

```
bsub -R
"cu[type=:pref=:maxcus=:usablecuslots=:balance:excl]"
```

usablecuslots=x

- LSF will use at least *x* slots in all except the last CU.
- With *usablecuslots*, LSF will try to pack job allocation as much as possible.

Balance the job's tasks across CUs

```
bsub -R
"cu[type=:pref=:maxcus=:usablecuslots=:balance:excl]"
```

- Slots allocated to the job are distributed evenly among CUs, using as few CUs as possible.
- *balanced allocation:* for any two CUs in the allocation, the difference between the number of slots occupied in each CU is at most 1.
- Typical Use Case:
  - Some applications are more efficient when its tasks are split evenly among the CUs occupied by the job.
  - *bsub –R "cu[balance]" ./app*

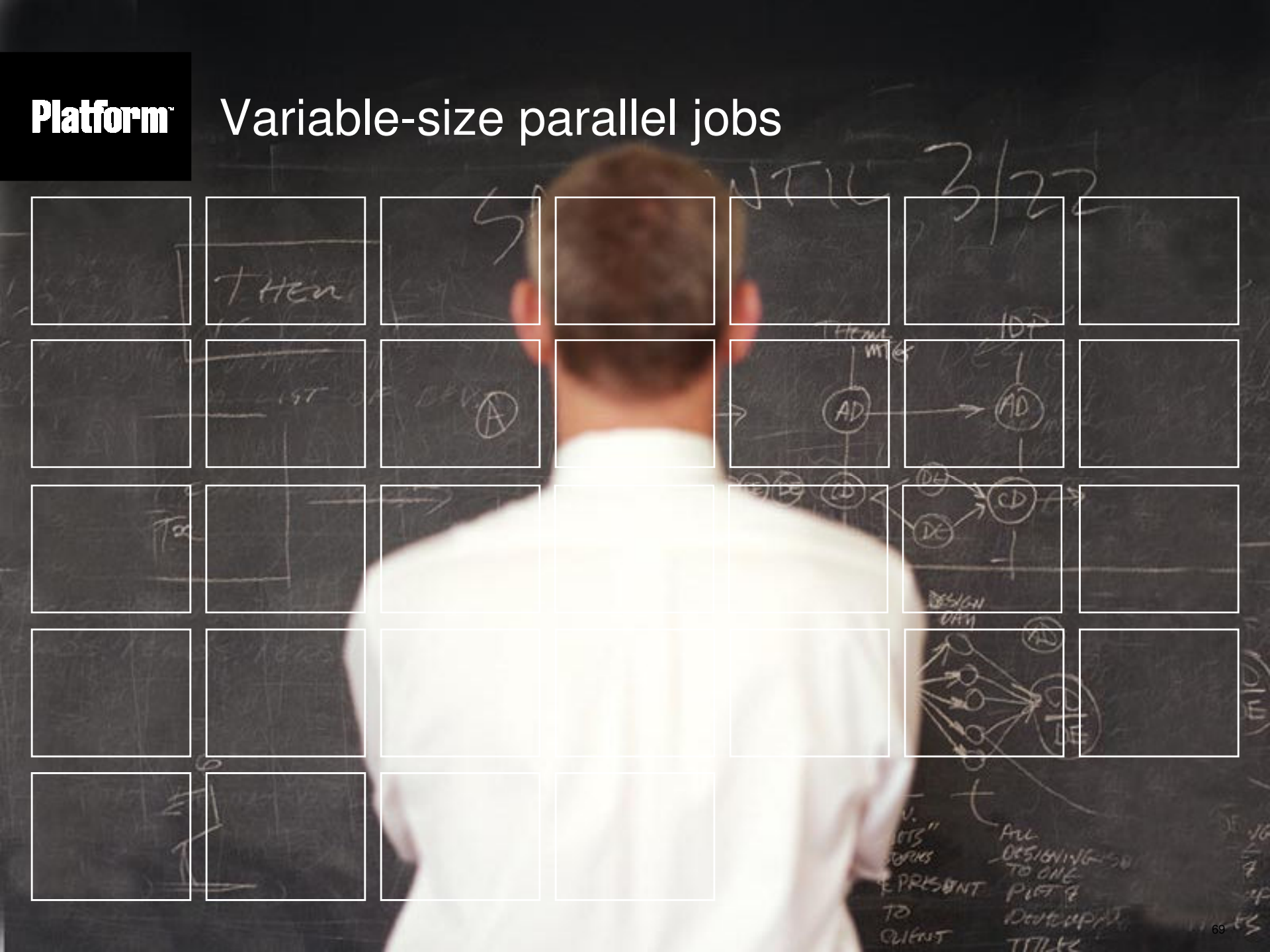
Exclusive use of CUs

```
bsub -R
"cu[type=:pref=:maxcus=:usablecuslots=:balance:excl]"
```

- A CU-level exclusive job can only start within a CU in which no other jobs are running.
- Once CU-level exclusive job starts, no other jobs are permitted to use the CUs.
- Typical use case:
  - exclusively occupy entire CUs to prevent other jobs sharing network bandwidth.
  - *bsub –R "cu[excl:type=enclosure]" ./app*

Platform™

Variable-size parallel jobs



Variable-size parallel jobs

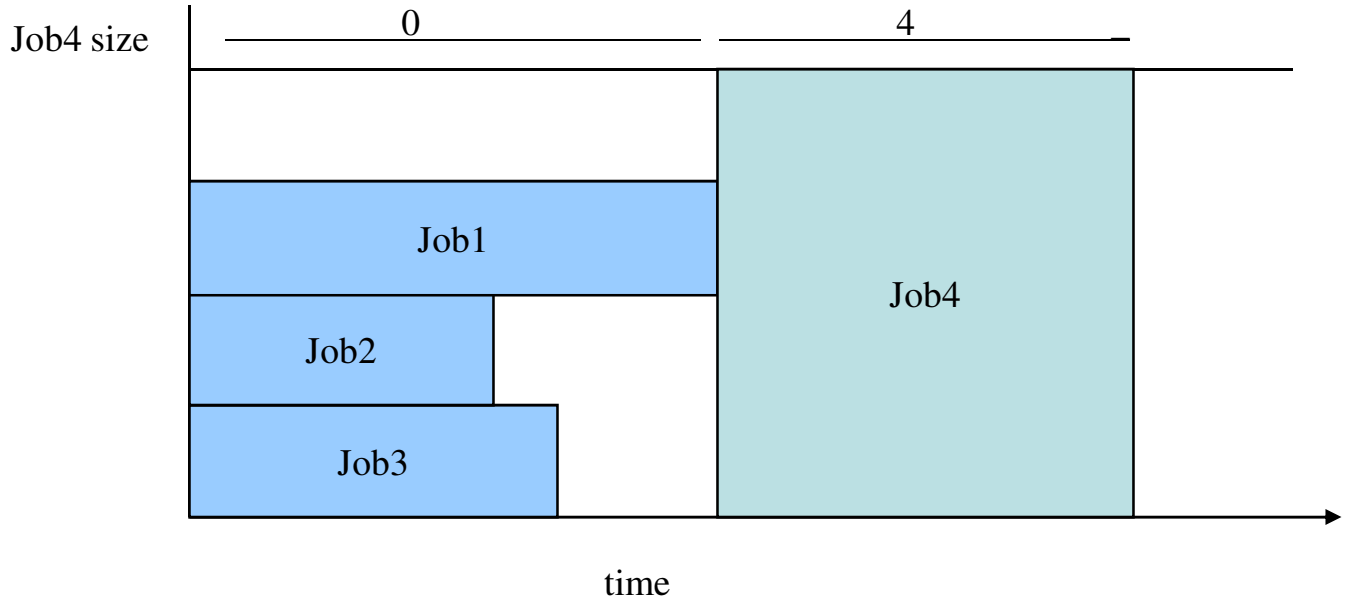
- Purpose of this feature: Replace static allocation for parallel jobs with dynamic allocation, i.e. allow jobs to gain and release slots as needed, thereby positively impacting cluster utilization.

- Aimed at embarassingly parallel workloads (parametric studies, montecarlo algorithms, etc..)

- Could be used by MPI applications but requires significant reworking of the application (and using MPI-2 features)

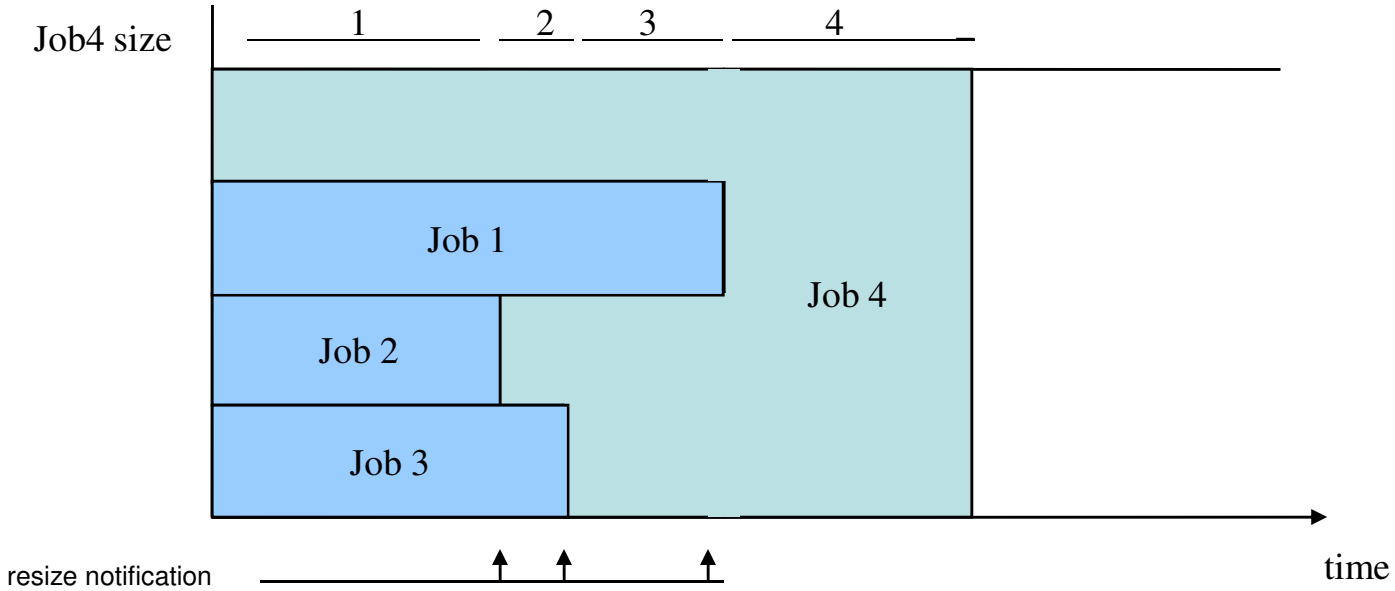
Why variable-size jobs

- Fixed job slot reservation may lead to resource wastage
- Backfill scheduling relies on runtime limits not always available, or accurate. .



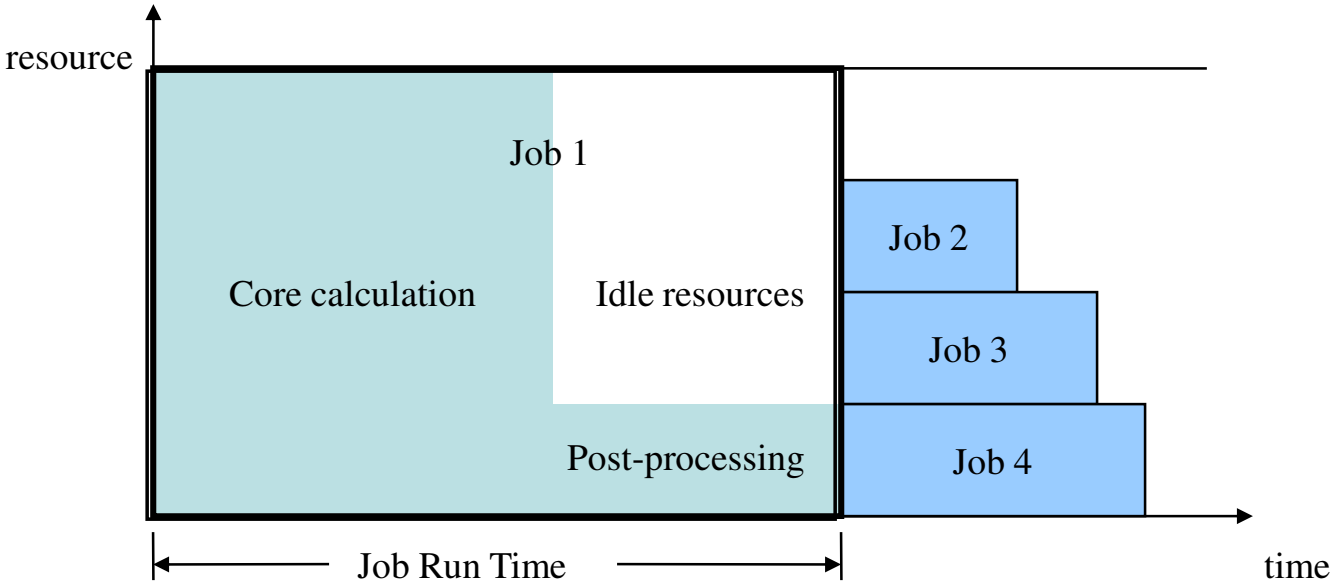
Autoresizable job

- Autoresizabel jobs have a variable number of slots
- An autoresizable job to starts as soon as it can acquire a minimum needed number of slots.
- Give job more slots as they become available, until job gets maximum needed slots.

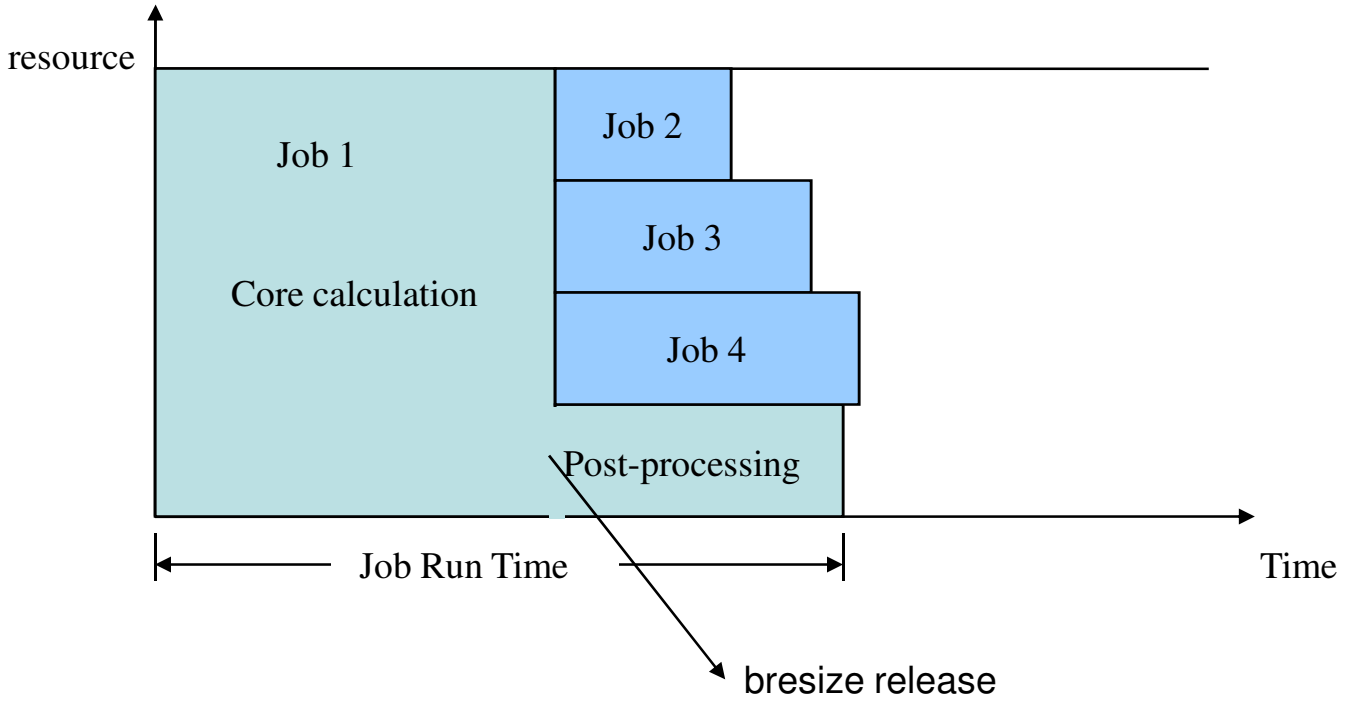


Long tail workloads

- Applications may have a sequential processes that run after the core calculation → wasted resources.
- Example: Jobs use 64 CPUs. After the calculation is finished, job performs some post-processing --- only 1 CPU needed (on first-execution host).



Long tail workloads with resizable jobs



Operations on resizable jobs

- Autoresizable jobs
  - Starts when min. required slots is available
  - Grows until it reaches max. required slots.
- Cancel a resize request
  - Stops an autoresizable job from growing.
- Release slots
  - Allows a running job to release slots.

Integrating resizing with the applications

- Can (optionally) configure a job so that a **resize notification command** runs whenever the job resizes.
  - runs on the first execution host.
  - purpose: to inform the application of the resize.
  - variables set in command's environment describe the new allocation
- **Grow:** execute command **after** LSF allocates slots.
- **Shrink:** execute command **before** LSF releases slots.
- Release can be triggered by either:
  1. the user; OR,
  2. the application

Submitting an autoresizable job

- bsub / bmod option **–ar** allows the user to set a job's autoresizable attribute.

*bsub **–ar** –n "3, 10" sleep 10000*

  - job will start with between MIN and MAX requested slots
  - while job has < MAX slots, job has resize request pending for the remaining slot

Cancelling an autoresizable job's requests

- Cancel an autoresizable job's resize request:
  - ***bresize cancel <jobID>***
  - After cancelling, the job will not get any additional slots.
  - Irreversible.

*bsub **–ar** –n 3,10 sleep 10000*

job gets 4 slots → request 1-6 additional slots

bresize cancel <jobId> → job gets no more slots

Releasing slots from a running job

- Release slots from a running job:
  - ***bresize release "3*hostA hostB" <jobID>***
  - job releases 3 slots on *hostA*, and all slots on *hostB*
  - jobs must maintain > 0 slots on first execution host
- Special shortcuts for the following:
  - release all slots except for 1 slot on first execution host.
  - release all hosts except for the first execution host.

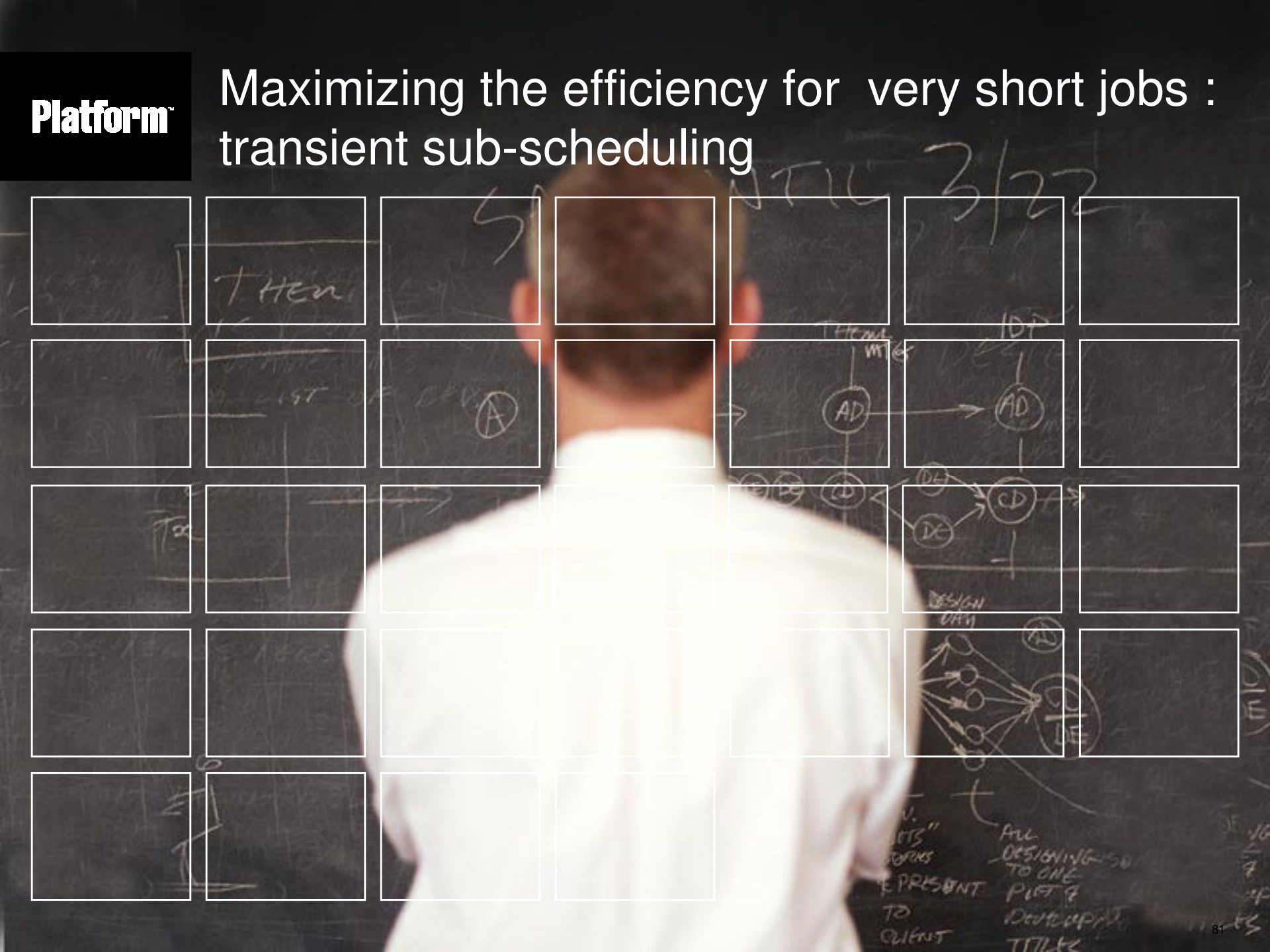
Benefits of resizable jobs

- More flexible resource allocation and better resources utilization (avoiding reserved slots)
- In conjunction with Session Scheduler, can be a valuable alternative to job arrays for embarrassingly parallel applications. Advantage: you can add topology constraints (same[]) to your job.

```
bsub –J "myjob[1-1000]" myapp
bsub  –n 1,100  -R "same[sanswitch] ssched –J "myjob[1-1000]" myapp
```

Platform™

Maximizing the efficiency for very short jobs : transient sub-scheduling



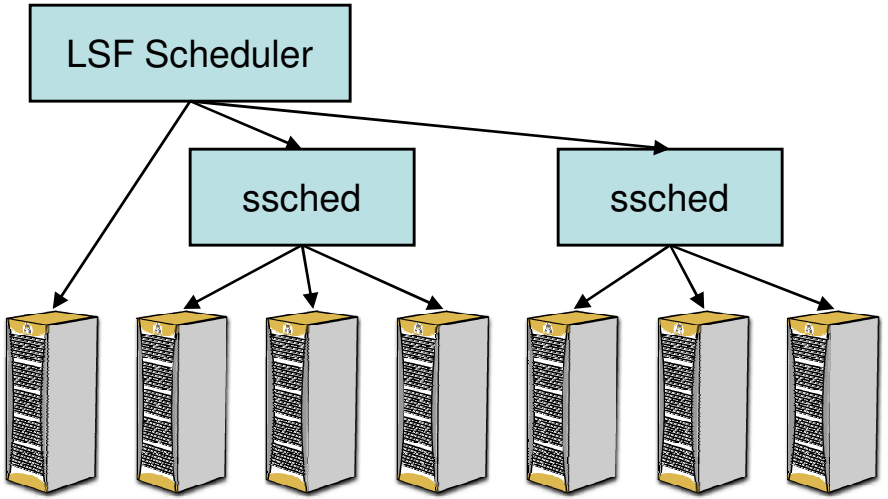
Platform

Scheduling Challenges with short jobs

- As clusters grow and the volume of workload increases, having the scheduler involved in every single scheduling decision can become an significant overhead.
  - Need to delegate scheduling decisions
- A large volume of short jobs can place a heavy load on the scheduler, which impacts overall system performance and cluster utilization.
  - Need to minimize the scheduling latency for short jobs.
- Tuning a single scheduler instance to give the best performance across a wide range of job types is challenging.
  - All existing polices must be honoured.

Introducing the LSF Session Scheduler

- The LSF Session Scheduler implements a hierarchal, personal scheduling paradigm that provides very low-latency execution.
- With very low latency per job, the Session Scheduler is ideal for executing very short jobs, whether they are a list of tasks, or job arrays/parametric execution.
- Each Session Scheduler is dynamically scheduled in a similar manner to a parallel job.
- Work is submitted as a job array or a task file
- Resource accounting is collected and appears in specific accounting files.



Platform

User Interface

- Usage:
  - *bsub –n <min>[,<max>] ssched [ssched_options] <task definition>*

- Task Definition
  - Command line
    - Specify individual task or task array
    - Example:

      ssched [task options] mytasks [arguments]

  - Task Definition File
    - ASCII file containing one line per task or task array
      - [task options] taskcommand [arguments]
    - Example:

      ssched [task options] –tasks <task_definition_file>

User Interface

- Task Options
  - *-J [index_list]* *(Task array)*
  - *-E "pre_exec_command [arguments ...]"* *(Pre execution)*
  - *-PE "post_exec_command [arguments ...]"* *(Post execution)*
  - *-B "starter [starter] [%USRCMD] [starter] "* *(Task Starter)*
  - *-i input_file* *(Input file)*
  - *-o out_file* *(Output file)*
  - *-e err_file* *(Error file)*
  - *-W [hour:]minute* *(Run limit)*
  - *-M mem_limit* *(Memory limit, KB by default)*
  - *-Q "exit_code ..."* *(Requeue exit values)*
- Input file, output file and error file support %J (jobid) %I (jobindex) %T (task id) and %X (task index) replacement.

Platform

User Interface

- Example 1:
  - *bsub –n 100 ssched –J “mytask[1-50000]” –E precmd –O postcmd mytask*
  - Description
    - Submit a Session Scheduler to run a task array with 50000 elements

- Example 2:
  - *bsub –n 100 ssched -tasks my.taskfile*
  - Contents of my.taskfile:
    - *-J [1-200] mytask*
    - *-J [1-100] -o my.out.%J.%I.%T.%X -e my.err.%J.%I.%T.%X hostname*
  - Description
    - Submit a Session Scheduler to run a list of tasks, which is defined in my.taskfile, where:
    - %J is job id
    - %I is job index
    - %T is task id
    - %X is task index

User Interface

- A snapshot of the running tasks is posted to the session job at a regular interval
  - Total number of tasks, number of pending tasks, number of running tasks, number of completed tasks, number of failed tasks
  - Use bjobs -l or bread to display

```
bread 1999
JOBID    MSG_ID FROM    POST_TIME      DESCRIPTION
1999     0      jchan   May 29 15:16   Tasks: RUN=1 PEND=0 DONE=10000 ERR=0
```

- A summary of all failed tasks is emailed to the user (or redirected to file)

Advantages

- Better intrinsic efficiency (local scheduling)
- Less load on the master -> better global efficiency
- Better control on turn-around times: the user can decide on how many cpus he wants is set of tasks to run; once allocated, the slots are dedicated to the job- no more competition with other jobs
- Works even better with variable-size jobs:

```
bsub -n 1-20 -ar ssched -J "mytask[1-50000]" -E precmd -O
   postcmd mytask
```

The job starts as soon as there is one slot and uses up to 20 slots. When the task list becomes empty, session scheduler releases slots as tasks finish.

.

Advantages

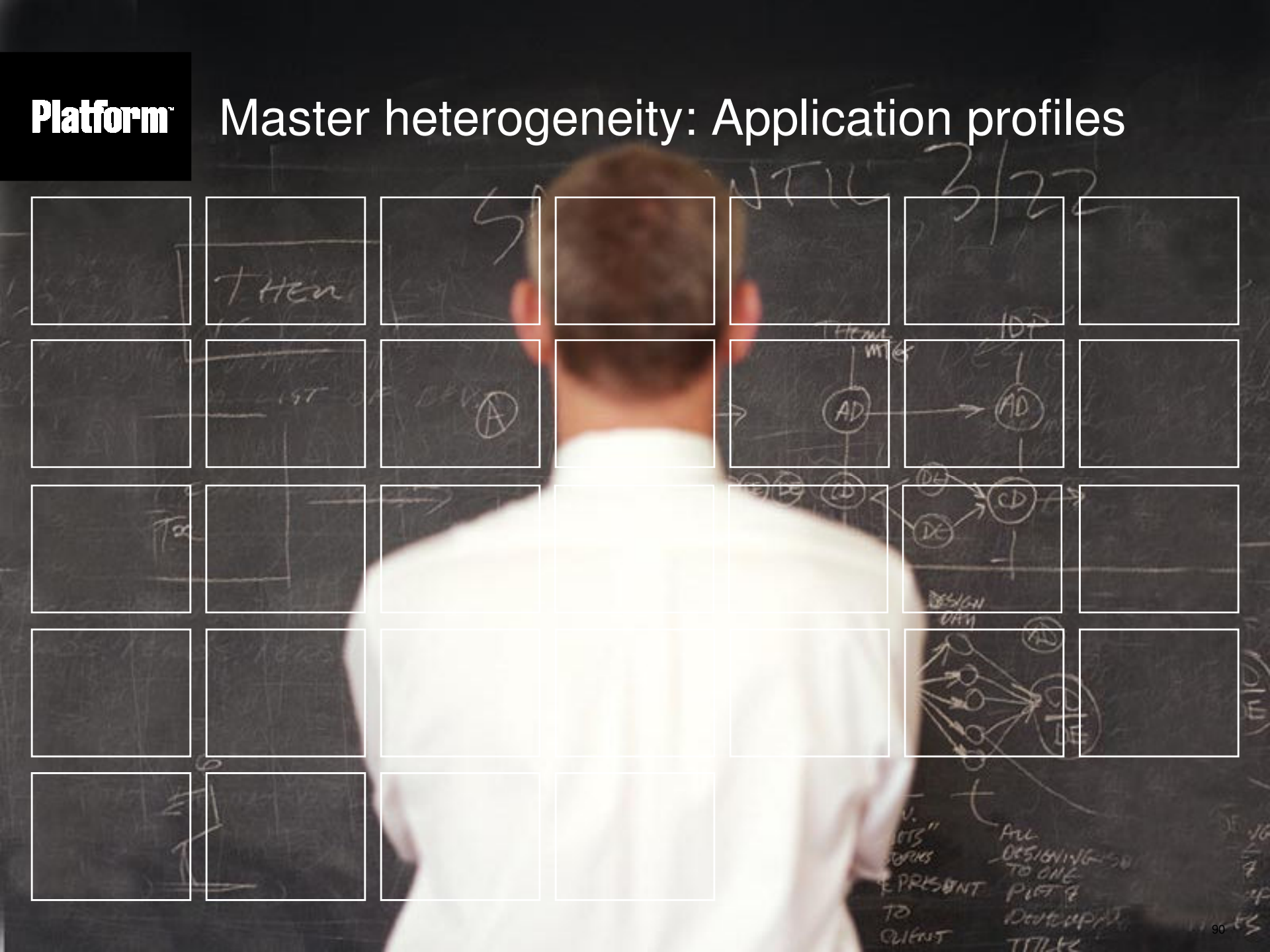
- Can be an alternative for Job Arrays even for not-so-short jobs, if topology constraints are needed

bsub –J "myjob[1-1000]" myapp

bsub –n 1,100 -R "same[enclosure] ssched –J "myjob[1-1000]" myapp

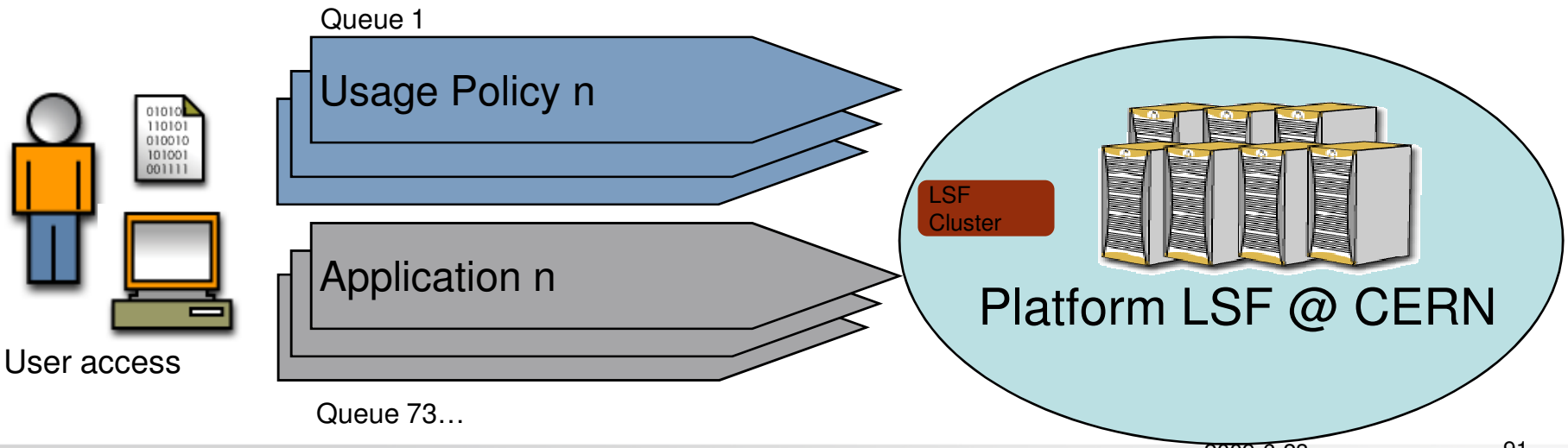
Platform

Master heterogeneity: Application profiles



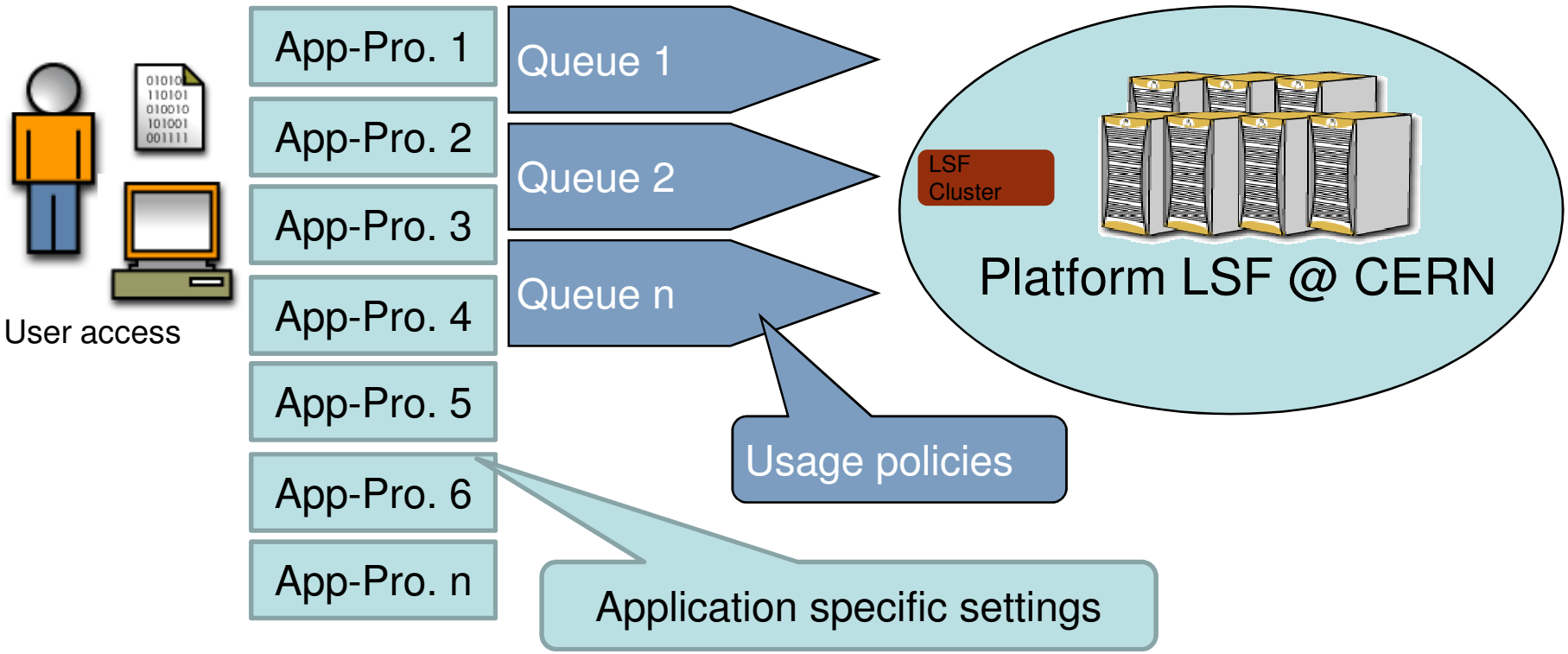
Master heterogeneity: Application profiles

- Managing many different requirements
  – Queues are not enough!
- Some facts about LSF queues:
  – Queues have been used to define usage-policies and application specific settings
  – There is not one (!) real queue in LSF – it's just a UI concept
  – Therefore you can have as many as you want – no performance penalty!
    - Could each user have his own queue? Yes, give him 7 if (s)he wants to…
    - But, this does not help manageability beyond certain numbers



Master heterogeneity: Application profiles

- Add an independent dimension – Application Profiles
  - Application profiles are used to define application specific settings
  - Leaves Queues for usage policies
  - Example: use 20 application profiles and 20 queues instead of 400 queues with combined settings



Platform

Application profile

- What options can we define in an application profile in 7.0?
  - pre-exec
  - post-exec
  - job starter
  - requeue exit values
  - limits
  - chunk job size
  - rerunnable
  - and more...
- Defined settings are applied to all jobs associated with this application profile

Application profile (cont)

- With application encapsulation, LSF supports 4 levels in terms of application behaviors:
  - Global: parameters defined in lsf.conf and lsb.params apply to all jobs
  - Queue: Queue level parameters apply to jobs submitted to that queue
  - Application: Application level parameters apply to jobs associated with applications
  - Job: Job level parameters only apply to the specific job

Creating an Application Profile

- $LSF_ENVDIR/lsbatch/<cluster_name>/configdir/lsb.applications

```
Begin Application
NAME = fluent
DESCRIPTION = Fluent Version 9
CPULIMIT = 24:0/hostA # 24 hours of host hostA
FILELIMIT = 20000
DATALIMIT = 20000 # jobs data segment limit
CORELIMIT = 20000
PROCLIMIT = 5 # job processor limit
REQUEUE_EXIT_VALUES = 55 34 78
End Application

Begin Application
NAME = lsdyna
DESCRIPTION = dyna version 10
PRE_EXEC=/user/share/dyna_pre.sh
POST_EXEC=/user/share/dyna_post.sh
PROCLIMIT = 4 # job processor limit
End Application
```

Using application profiles

- Submit a job associated with an application profile
  - Submit a app1 job
    - bsub –app app1 –q overnight a.out
  - Submit a app2 job to the same queue
    - bsub –app app2 –q overnight b.out
- List all app2 jobs
  - bjobs –app app2
- List all defined application profiles

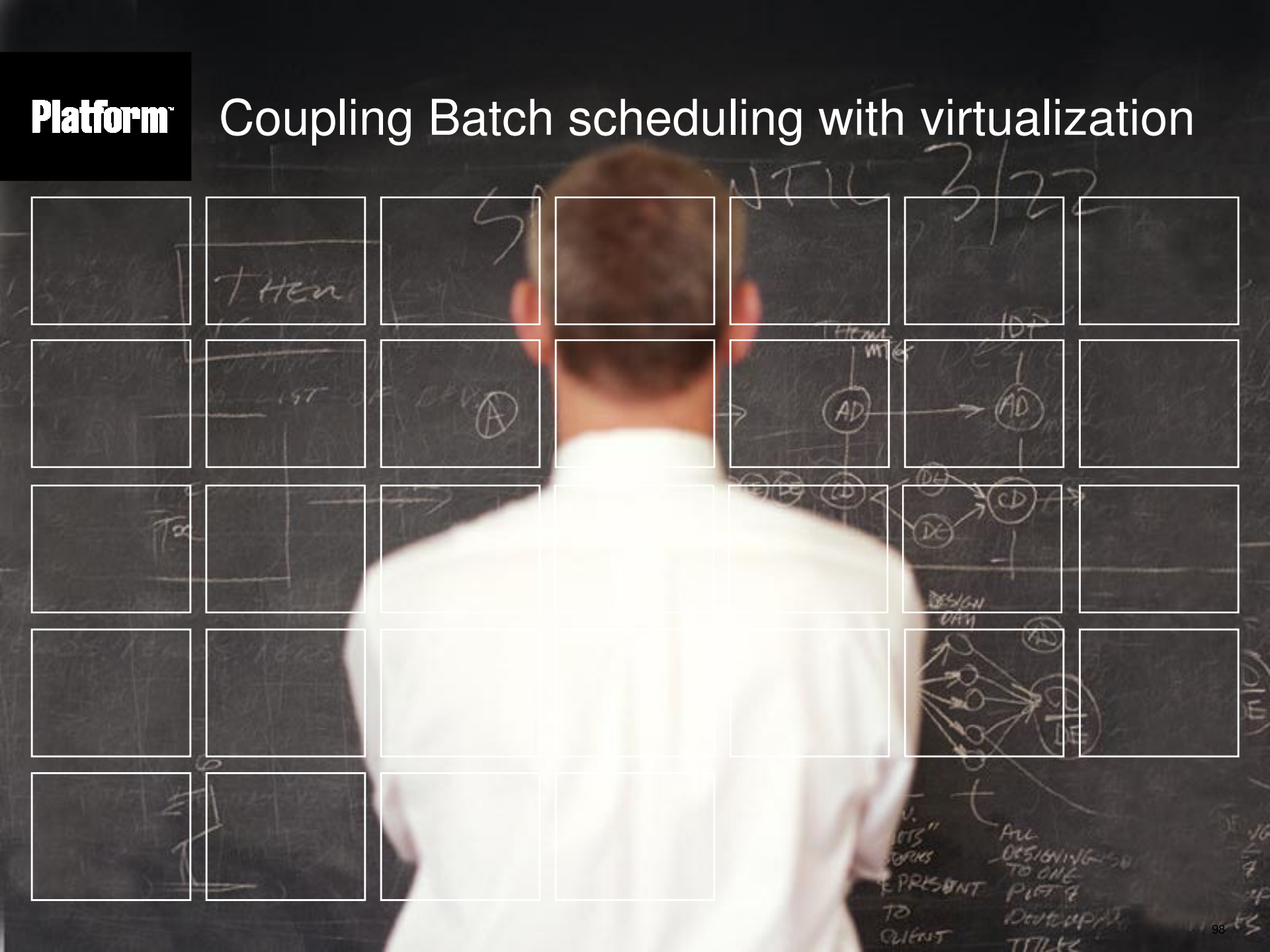
```
% bapp
APPLICATION_NAME              NJOBS  PEND   RUN  SUSP
app1                            300   155   145     0
app2                             25     0    25     0
```

Benefits of appication encapsulation

- Reduces the number of queues
- Simplifies user interface: -app replaces many options
- Makes fine-grained options (REQUEUE_EXIT_VALUES, JOB_CONTROLS,RES_REQ,RERUNNABLE etc..) compatible with an heterogeneous workload
- Helps to enforce scheduling options (resource requirements…)

Platform

Coupling Batch scheduling with virtualization

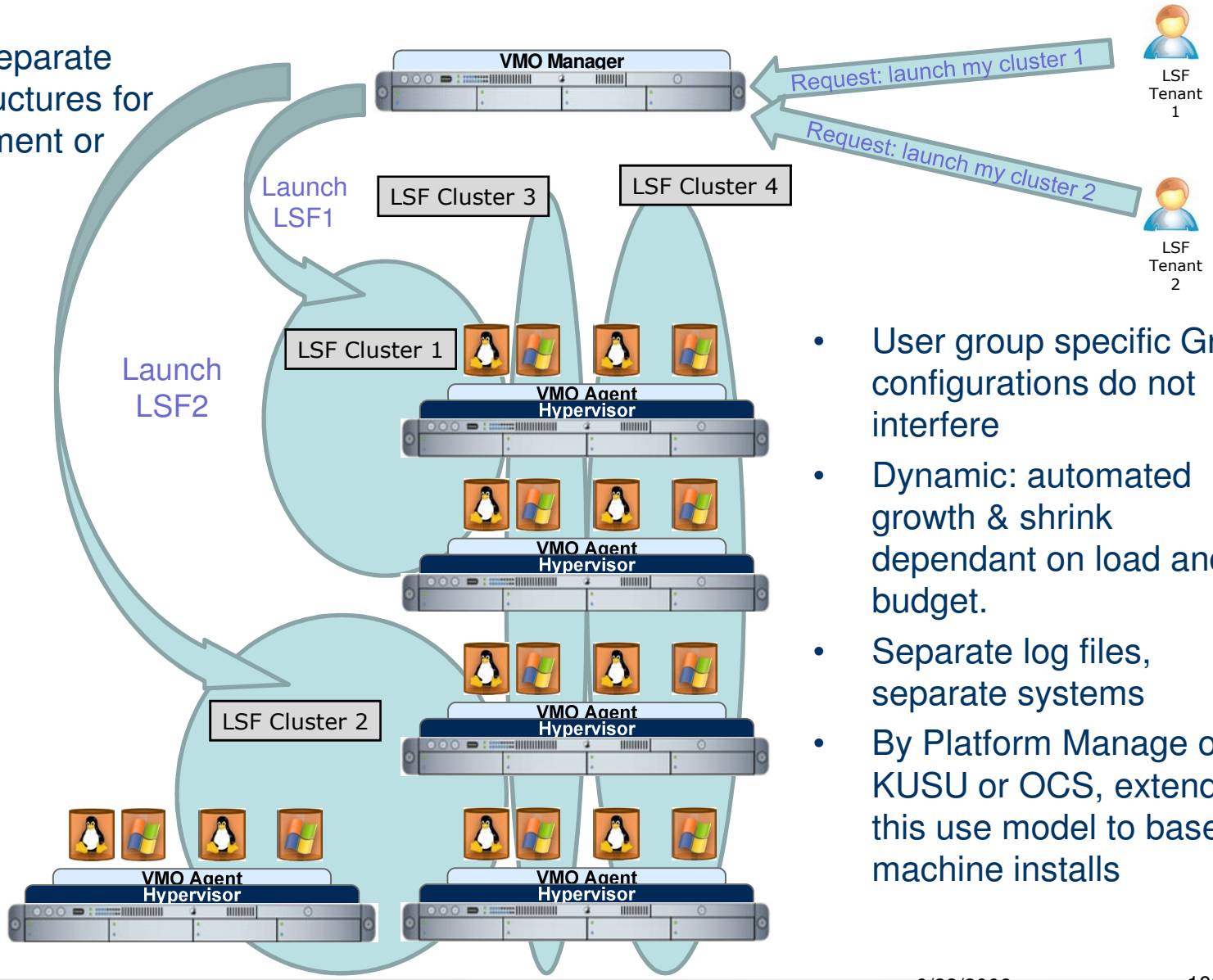


Coupling Batch scheduling with virtualization

- Combinations of Virtualization with Grid Technology
  - Dynamic inclusion of VMs in the batch cluster
    - You need SLC4.3 instead of SLC5.2 ? Get the correct OS/patch level / config / preinstalled apps /… automatically as VM
  - Dynamic “personality” change of compute nodes
    - Best possible computational performance by automated real machine (RM) deployment
  - Virtually private Grids within (and beyond) shared resource pools
    - See next slide
- These topics are related to the talk of Dr. Sebastien Goasguen.

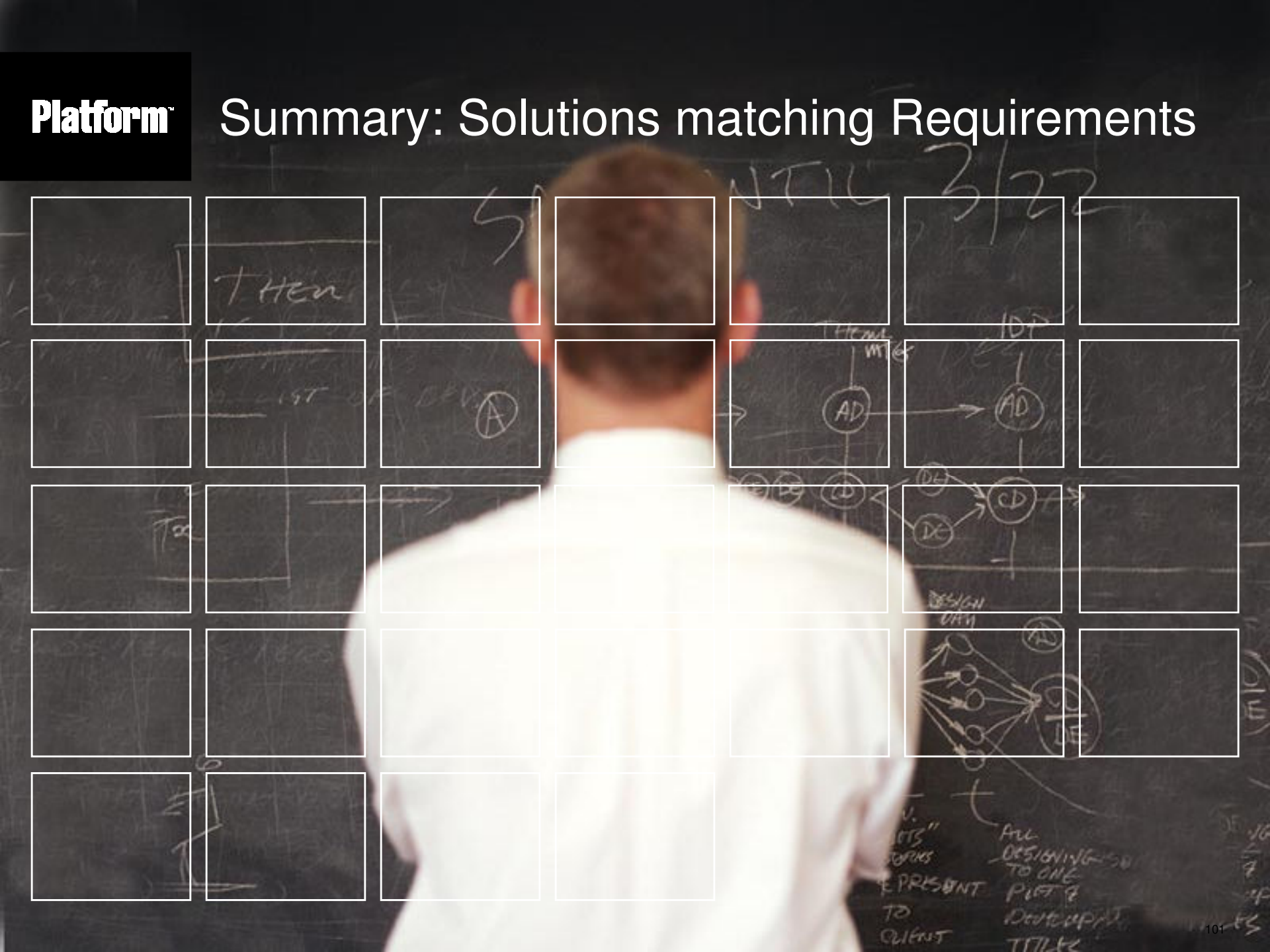
Multi-Tenant Grid Infrastructure on demand

- Instantiate separate Grid infrastructures for each experiment or user group



- User group specific Grid configurations do not interfere
- Dynamic: automated growth & shrink dependant on load and budget.
- Separate log files, separate systems
- By Platform Manage or KUSU or OCS, extend this use model to base machine installs

Summary: Solutions matching Requirements



Summary: solutions matching requirements

- Several properties and functionalities of Platform LSF are used in conjunction to assure
  - Responsiveness to the user
  - Best possible management of workload
  - Best possible management of resources

Beyond the mature classical concept, alternatives are:

- Combinations of Virtualization with Grid Technology
  - Dynamic inclusion of VMs in the batch cluster
    - Serve more different resource requirements
    - Isolation and protection of workload
  - Dynamic “personality” change of compute nodes
  - Virtually private Grids

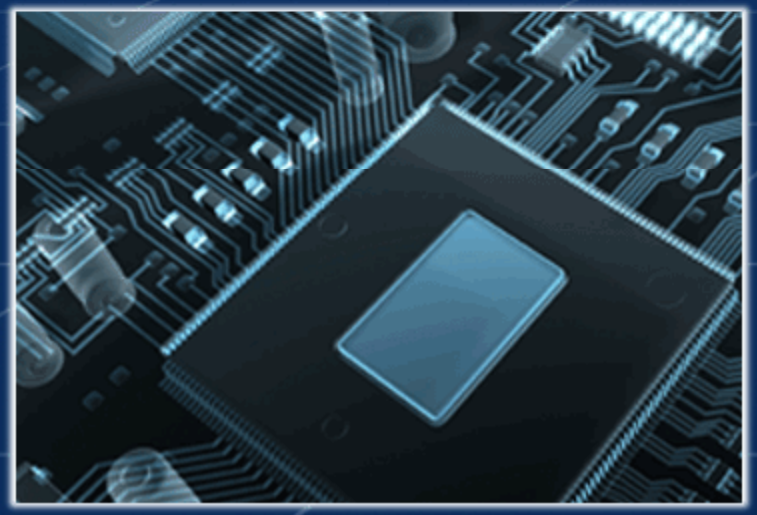
Platform™

Powering High Performance

Q & A

www.platform.com

Platform™



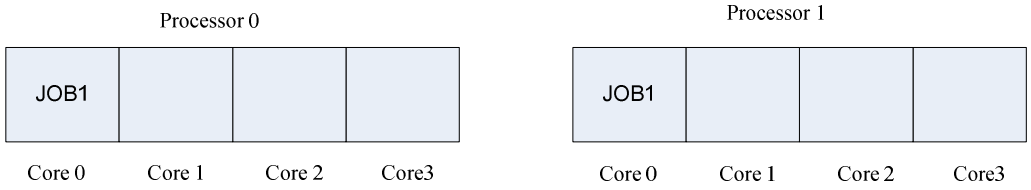
Details & Examples on Processor Binding

LSF_BIND_JOB=BALANCE

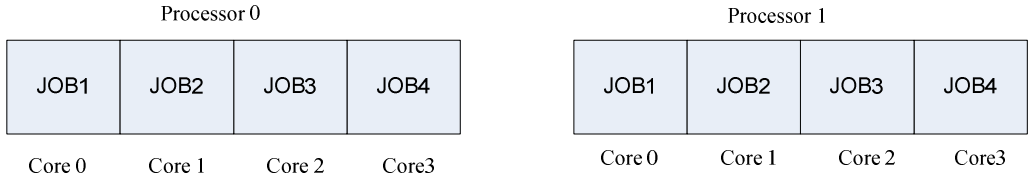
- Equivalent to the existing BIND_JOB=Y where processes are load balanced across the available processors/cores/threads when the job is a sequential job.
  - If the PCT level is processor level, the least loaded physical processor will be selected
  - If the PCT level is core level, the least loaded core on the least loaded physical processor will be selected
  - If the PCT level is thread level, the least loaded thread on the least loaded core on the least loaded physical processor will be selected

Example for BALANCE

- Suppose there is a single 2 processor quad core host.
  When a parallel job is submitted with –n 2 –R"span[hosts=1]", the job will be bound to the first core on the first processor and the first core on the second processor:



- Another 3 jobs are submitted with –n 2 –R"span[hosts=1]":



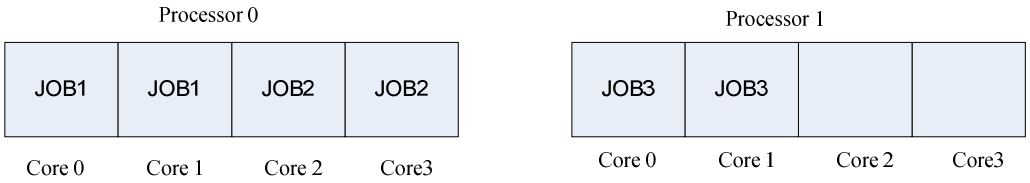
LSF_BIND_JOB=PACK

- LSF tries and packs all the processes onto a single processor.
  If this cannot be done, LSF tries to use as less processors as possible meanwhile an email is sent to the job's user when the job is dispatched and also finished.
- If there is no free processor/core/thread left, LSF tries to use BALANCE for new jobs.

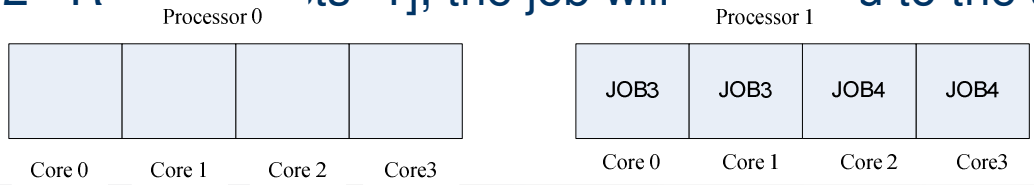
LSF_BIND_JOB=PACK (cont.)

Example for PACK

- 3 single-host parallel jobs are submitted with –n 2 –R”span[hosts=1]” when the PCT level is core level.
- The first job (JOB1) and the second job (JOB2) will be packed to the first processor.
- the third job (JOB3) will be bound to the second processor (while not oversubscribe the cores on the first processor) :



- After JOB1 and JOB2 finished,
  when a single-host parallel job (JOB4) is submitted with
  –n 2 –R”span[hosts=1], the job will be bound to the second processor:



LSF_BIND_JOB=ANY

- LSF tries to bind a job to the first N available processors/cores/threads with no regard for locality.
- If the PCT level is core, LSF binds the first N available cores regardless of whether they are on the same processor or not. LSF will arrange the order based on APIC ID.

APIC ID: Advanced Programmable Interface Controller ID

LSF_BIND_JOB=USER

- LSF will bind the job to the value of **$LSB_USER_BIND_JOB** specified in the user submission environment
- This allows the Administrator to delegate processor binding policies to the job submission users.
- **LSB_USER_BIND_JOB = Y| N | NONE | BALANCE | PACK | ANY**
  'ANY' is used for invalid values

LSF_BIND_JOB=USER_CPU_LIST

- LSF tries to bind a job to the CPUs specified in the job submission environment variable **$LSB_USER_BIND_CPU_LIST**

- LSF does not check if the value is valid for the execution host.
  It is the user's responsibility to specify the adequate CPU list of the execution host.

- $LSB_USER_BIND_CPU_LIST may contain multiple CPUs, separated by comma, and/or ranges.
  For example, 0,5,7,9-11.

Env variables on execution env

LSF sets the environment variables in the job execution environment:

1. $LSB_BIND_JOB the value of binding option is set.
   When the binding option is USER, LSB_BIND_JOB is set to the actual binding option specified by the user.
   if LSF_BIND_JOB=Y, LSB_BIND_JOB is set to BALANCE.
   If LSF_BIND_JOB=N, LSB_BIND_JOB is set to NONE.
2. $LSB_BIND_CPU_LIST
   the actual CPU list is set for sequential jobs, single host parallel jobs.
   For multi-host parallel jobs,
   - LSB_BIND_CPU_LIST is set to the value in submission environment variable
3. $LSB_USER_BIND_CPU_LIST
   - LSF_BIND_CPU_LIST is set to empty string if there is no environment variable
   $LSB_USER_BIND_CPU_LISET specified in submission environment

Examples

(1) BALANCE for sequential job

Job <834>, User <gwpang>, Project <default>, Status <RUN>, Queue <normal>, Command <sleep 300>
Thu Jun 19 02:54:19: Submitted from host <delpe05.lsf.platform.com>, CWD <$HOME/scratch>;
Thu Jun 19 02:54:24: Started on <delpe05.lsf.platform.com>, Execution Home </home/gwpang>,
Execution CWD </home/gwpang/scratch>;
Thu Jun 19 02:54:27: Resource usage collected.
MEM: 2 Mbytes; SWAP: 16 Mbytes; NTHREAD: 1
PGID: 14280; PIDs: 14280

Job <835>, User <gwpang>, Project <default>, Status <RUN>, Queue <normal>, Command <sleep 299>
Thu Jun 19 02:54:30: Submitted from host <delpe05.lsf.platform.com>, CWD <$HOME/scratch>;
Thu Jun 19 02:54:34: Started on <delpe05.lsf.platform.com>, Execution Home </home/gwpang>,
Execution CWD </home/gwpang/scratch>;
Thu Jun 19 02:54:37: Resource usage collected.
MEM: 2 Mbytes; SWAP: 16 Mbytes; NTHREAD: 1
PGID: 14331; PIDs: 14331

P level

gwpang@delpe05-102: taskset -cp 14280
pid 14280's current affinity list: 0,2,4,6
gwpang@delpe05-103:
gwpang@delpe05-103: taskset -cp 14331
pid 14331's current affinity list: 1,3,5,7

Examples (cont.)

C level

```
 gwpang@delpe05-141: taskset -cp 22508
pid 22508's current affinity list: 0
 gwpang@delpe05-142:
 gwpang@delpe05-142: taskset -cp 22534
pid 22534's current affinity list: 1
 gwpang@delpe05-143:
 gwpang@delpe05-143: taskset -cp 22601
pid 22601's current affinity list: 2
 gwpang@delpe05-144:
 gwpang@delpe05-144: taskset -cp 22602
pid 22602's current affinity list: 3
```

T level

```
 gwpang@delpe07-139: taskset -cp 31774
pid 31774's current affinity list: 0
 gwpang@delpe07-140: taskset -cp 31775
pid 31775's current affinity list: 1
 gwpang@delpe07-141: taskset -cp 31794
pid 31794's current affinity list: 2
 gwpang@delpe07-142: taskset -cp 31798
pid 31798's current affinity list: 3
 gwpang@delpe07-143: taskset -cp 31799
pid 31799's current affinity list: 4
 gwpang@delpe07-144: taskset -cp 31802
pid 31802's current affinity list: 6
```

Examples (cont.)

(2) BALANCE for single host parallel job

Thu Jun 19 03:01:27: Submitted from host <delpe05.lsf.platform.com>, CWD <$HOME/scratch>, 2 Processors Requested;
Thu Jun 19 03:01:30: Started on 2 Hosts/Processors <delpe05.lsf.platform.com> <delpe05.lsf.platform.com>, Execution Home </home/gwpang>, Execution CWD </home/gwpang/scratch>;
Thu Jun 19 03:01:33: Resource usage collected.
MEM: 2 Mbytes; SWAP: 16 Mbytes; NTHREAD: 1
PGID: 16195; PIDs: 16195

Thu Jun 19 03:01:38: Submitted from host <delpe05.lsf.platform.com>, CWD <$HOME/scratch>, 2 Processors Requested;
Thu Jun 19 03:01:40: Started on 2 Hosts/Processors <delpe05.lsf.platform.com> <delpe05.lsf.platform.com>, Execution Home </home/gwpang>, Execution CWD </home/gwpang/scratch>;
Thu Jun 19 03:01:43: Resource usage collected.
MEM: 2 Mbytes; SWAP: 16 Mbytes; NTHREAD: 1
PGID: 16245; PIDs: 16245

P level

gwpang@delpe05-127: taskset -cp 16195
pid 16195's current affinity list: 0-7
gwpang@delpe05-128:
gwpang@delpe05-128: taskset -cp 16245
pid 16245's current affinity list: 0-7

Examples (cont.)

C level

```
 gwpang@delpe05-225: taskset -cp 29840
pid 29840's current affinity list: 0-2
 gwpang@delpe05-226: taskset -cp 29864
pid 29864's current affinity list: 3-5
 gwpang@delpe05-227:
```

T level

```
  gwpang@delpe07-105: taskset -cp 31142
pid 31142's current affinity list: 3,4,6
 gwpang@delpe07-106:
 gwpang@delpe07-106: taskset -cp 31158
pid 31158's current affinity list: 0,5,7
 gwpang@delpe07-107:
```