Adaptive COMPUTING

Intelligent Workload Management Software for HPC and Cloud Environments

Moab and TORQUE Highlights
CUG 2015

David Beer 28 Apr 2015
TORQUE Architect

Gary D. Brown
HPC Product Manager

Agenda

- **NUMA-aware Heterogeneous Jobs**
- **"Ascent" Project**
- **Power Management and Energy Accounting**
- **DataWarp Integration**
- **Nitro HTC Scheduler**
- **Collaboration Invitation**

NUMA-aware Heterogeneous Job Task Placement

NUMA-aware Job Task Placement

- **Moab job submission options (msub)**
  - Task-oriented (`tasks=nnn`), not job- or node-oriented
    - Logical processors/task (`lprocs=nn`)
    - Logical processor definition
      - Cores (`usecores` – on request)
      - Threads (`usethreads` – on request)
      - Cores or threads (`allowthreads` – default)
    - Memory per task (`memory=xxx`)
    - Other options per task (features, generic resources, accelerators, etc)
- **TORQUE qsub supports same syntax**

NUMA-aware Job Task Placement (continued)

- **Places task exclusively on resources of specified locality**
  - node *Compute node*
  - socket
  - numa *NUMA node (Intel Xeon "Haswell" and AMD Opteron)*
  - core
  - thread
- **Uses control groups (cgroups) to pin task to resources**
  - Processors
  - Memory
  - PCIe devices (GPUs, MICs)

NUMA-aware Job Task Placement (continued)

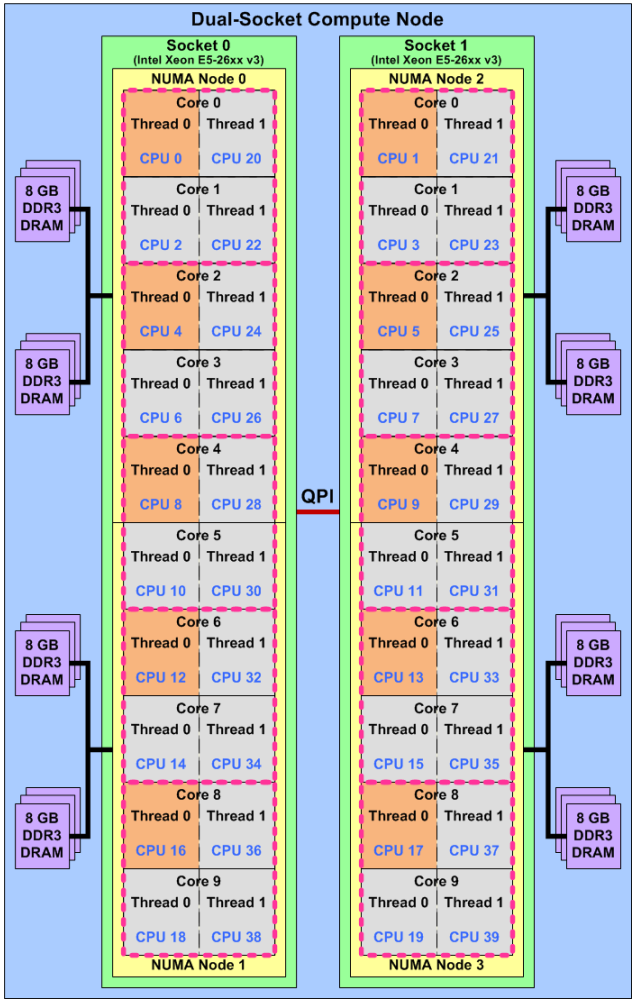
- **Example job submission commands**

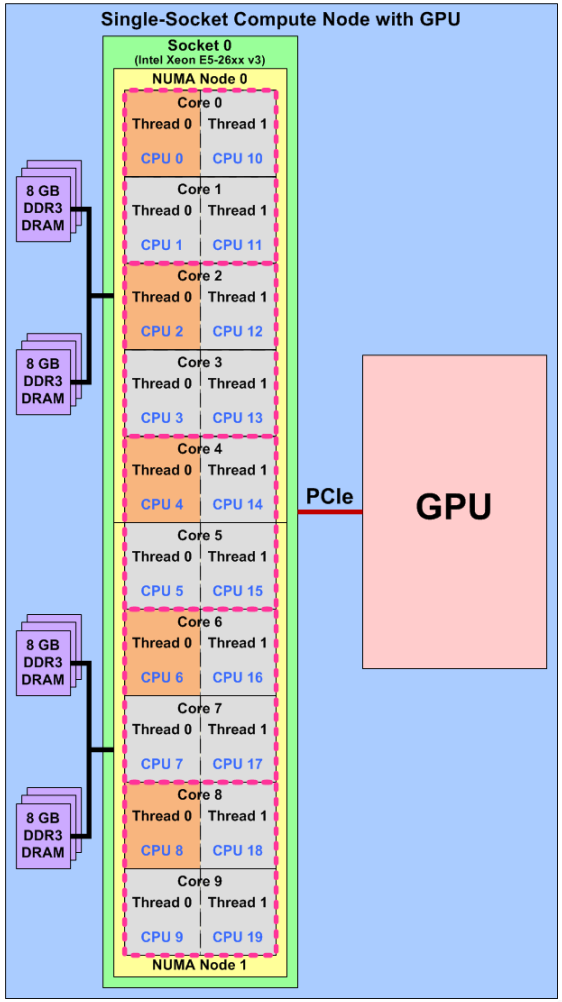
```
# Simple 1 task per core, regardless if hyper-threaded
qsub -L tasks=100:lprocs=1:usecores:place=core

# Coupled-simulation (1 solid task, 2000 Cray fluid tasks)
msub -L tasks=1:usecores:memory=800GB:place=node:bigmem
     -L tasks=2000:usecores:place=core:memory=2GB:cray

# 1 task using 1 thread per 2 threads (e.g., Intel MIC KNL)
qsub -L tasks=1200:lprocs=1:usethreads:place=thread=2

# 1 task per 2 sockets with 6 cores (3 / socket) and 1 GPU
msub -L tasks=100:lprocs=6:usecores:memory=200GB:gpu=1:
         place=socket=2
```

NUMA Example 1 – Heterogeneous Nodes

```
qsub –L tasks=288:lprocs=1:usecores:place=core=2 myMPIjob
```

10 tasks per node



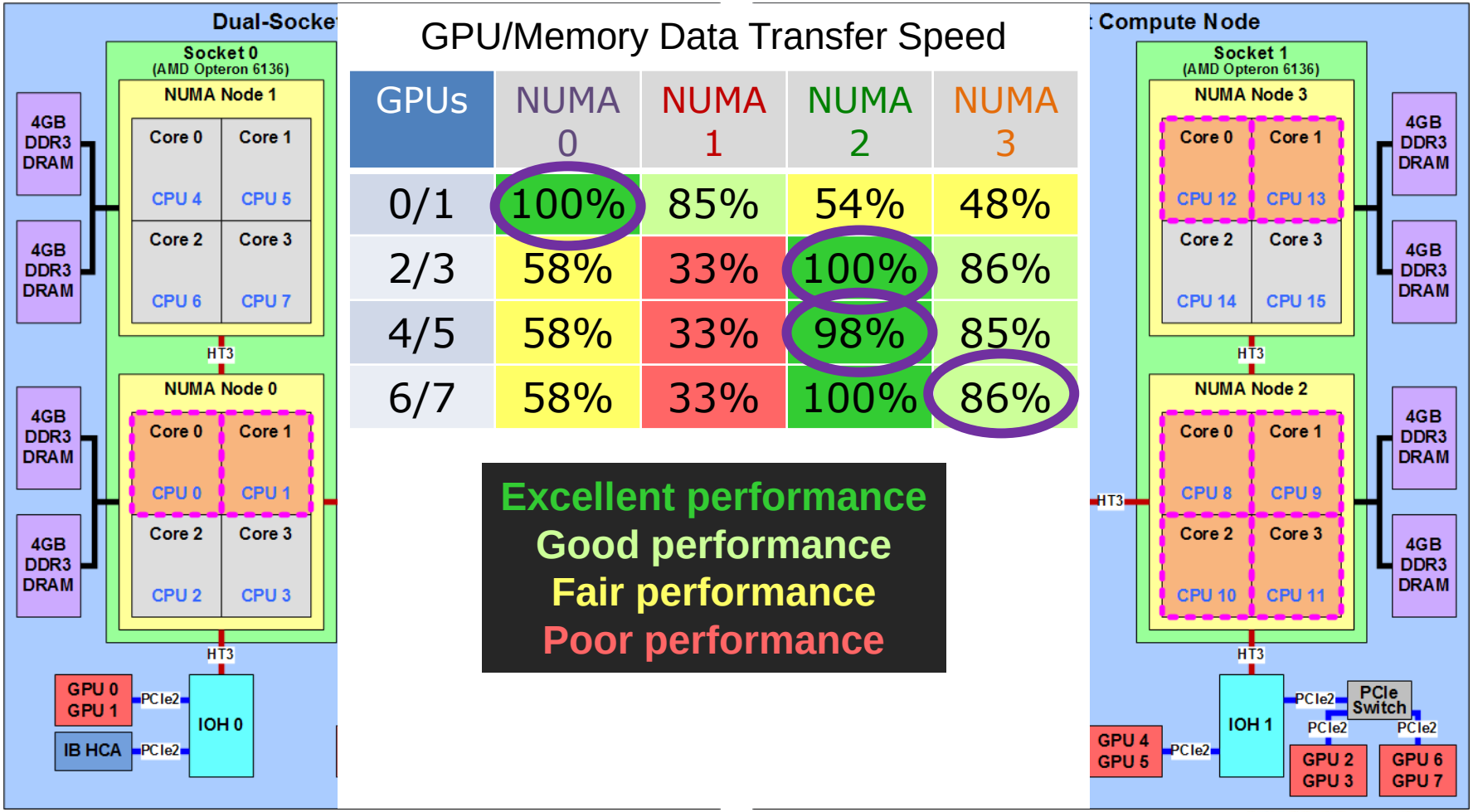


5 tasks per node

Adaptive COMPUTING

NUMA Example 2 – GPU Cluster

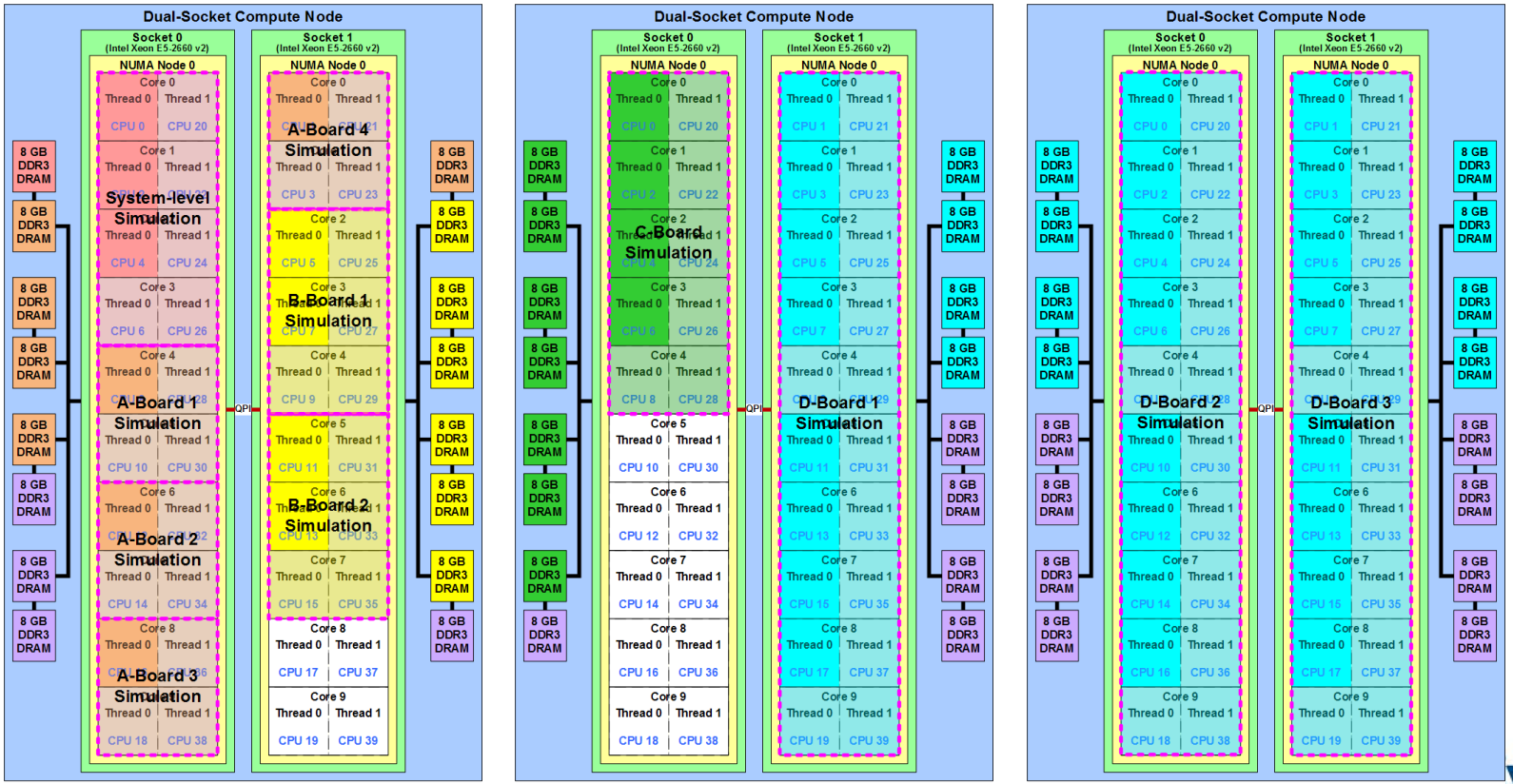
```
qsub -L tasks=16:lprocs=1:usecores:gpu=1:place=core myGPUsim
```



Adaptive COMPUTING

NUMA Example 3 – Avionics Simulation

```
msub -L tasks=1:lprocs=3:memory=5GiB:usecores:place=core=4   # red     System board
     -L tasks=4:lprocs=1:memory=10GiB:usecores:place=core=2  # orange  A board
     -L tasks=2:lprocs=2:memory=20GiB:usecores:place=core=3  # yellow  B board
     -L tasks=1:lprocs=4:memory=50GiB:usecores:place=core=5  # green   C board
     -L tasks=3:lprocs=8:memory=30GiB:usecores:place=socket  # blue    D board
     -l naccesspolicy=singlejob AvionicSim.pbs
```



NUMA-aware Benefits

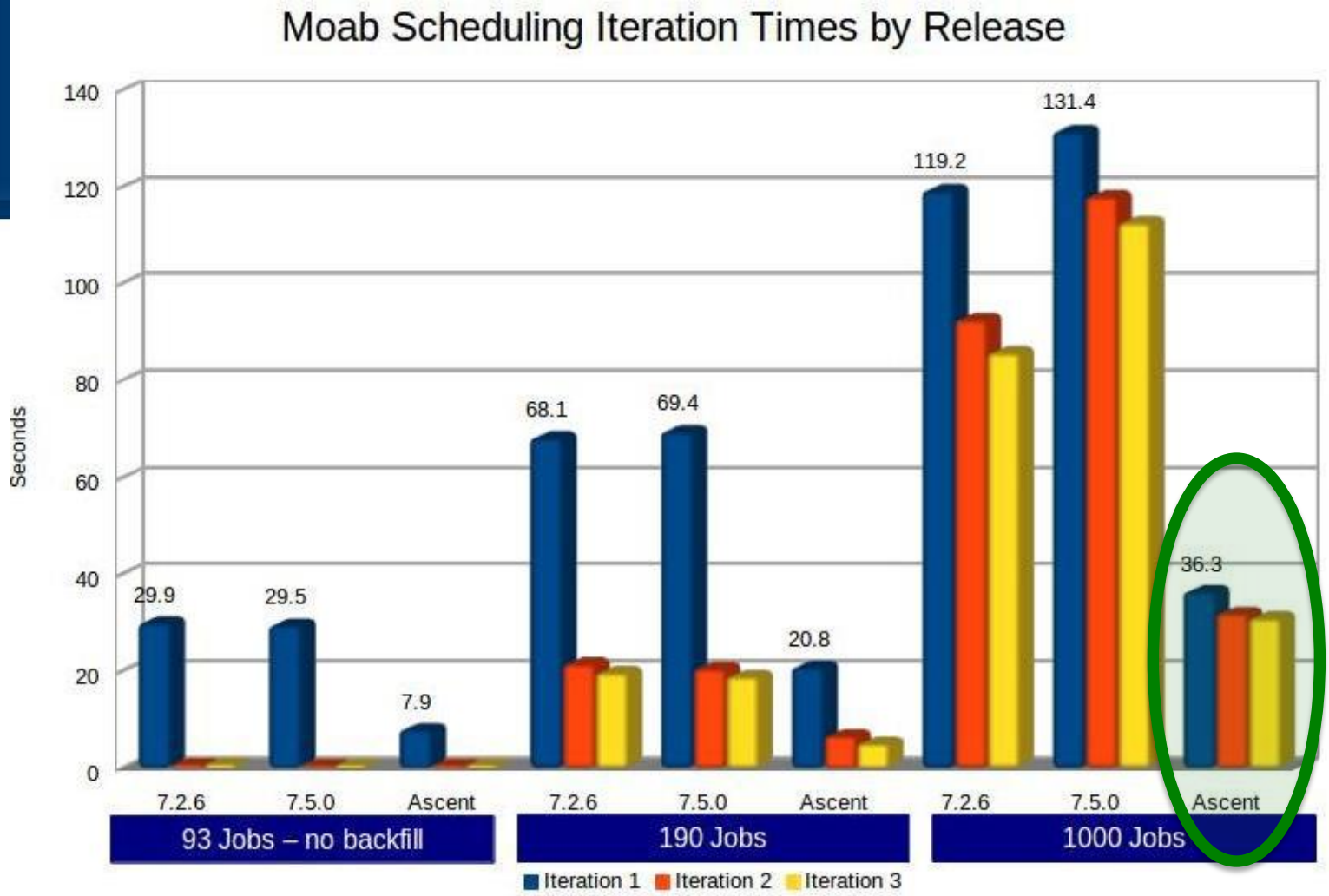
- **Accurately place jobs on cores according to memory access**
  - NUMA-optimized, memory-bound workloads can run up to 2.5x faster
  - More consistent job runtimes with predictable memory access times
  - Much less job-to-job or task-to-task interference on each compute node
- **User can define resources as needed by apps**
  - Request resources on a per-task type basis, allowing jobs to reserve only what they need
- **Increases hardware ROI with higher job throughput due to shorter job runtimes**

Ascent Project

Focus and Goals

- **Main goals**
  - Improve scalability (>10x)
  - Improve job throughput (>10x)
  - Improve stability (rock-solid)
  - Improve architecture
- **Timeframe**
  - 24 months with four deliveries
    - Moab 8.0 / TORQUE 5.0
    - Moab 8.1 / TORQUE 5.1
    - Moab 9.0 / TORQUE 6.0
    - Next version
- **People**
  - Focused team of top engineers

Moab 8.0 Example



Cray simulation using native resource manager and:
- 25,776 nodes including 64 login nodes
- Reservation depth of 100
- Reservation on 8564 nodes

Moab running on 8 core Intel Xeon L5520 2.27GHz with HyperThreading disabled and 60GB Memory, LOGLEVEL=3

Adaptive COMPUTING

TORQUE 5.0 Results

- **Reduced "jitter" by eliminating polling of compute nodes for job information**
- **Job submission rate of 250+ jobs per second for 100K jobs with no errors**

Moab 8.1 Results

- **Moab scheduling cycle time reduced by 55%**
- **Moab client command response delay reduced by 84%**
- **Moab job submission time reduced by 60%**

Moab 9.0 Results (target of 29 July)

- **Moab "query" client commands' response delay reduced to sub-second**

Power Management

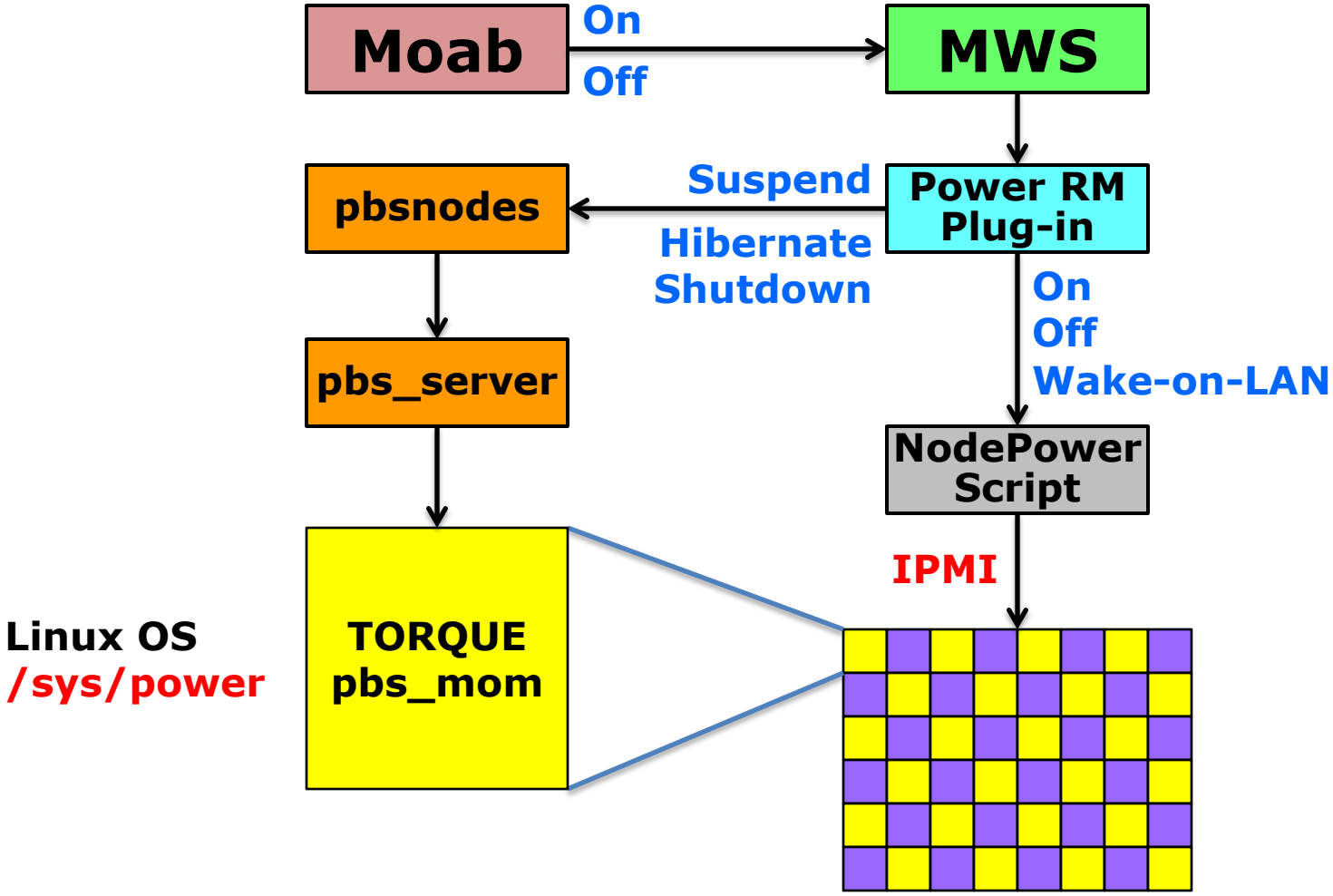
CPU Clock Frequency Control

- **Power management for running nodes**
- **Job submission cpuclock= option**
  - Absolute Clock Frequency Number
    - `cpuclock=2200`
    - `cpuclock=1800mhz`
  - Linux Power Governor Policy
    - `cpuclock=conservative`
  - Relative P-state Number
    - `cpuclock=0`
    - `cpuclock=p2`
- **Can set in job scripts and Moab templates**

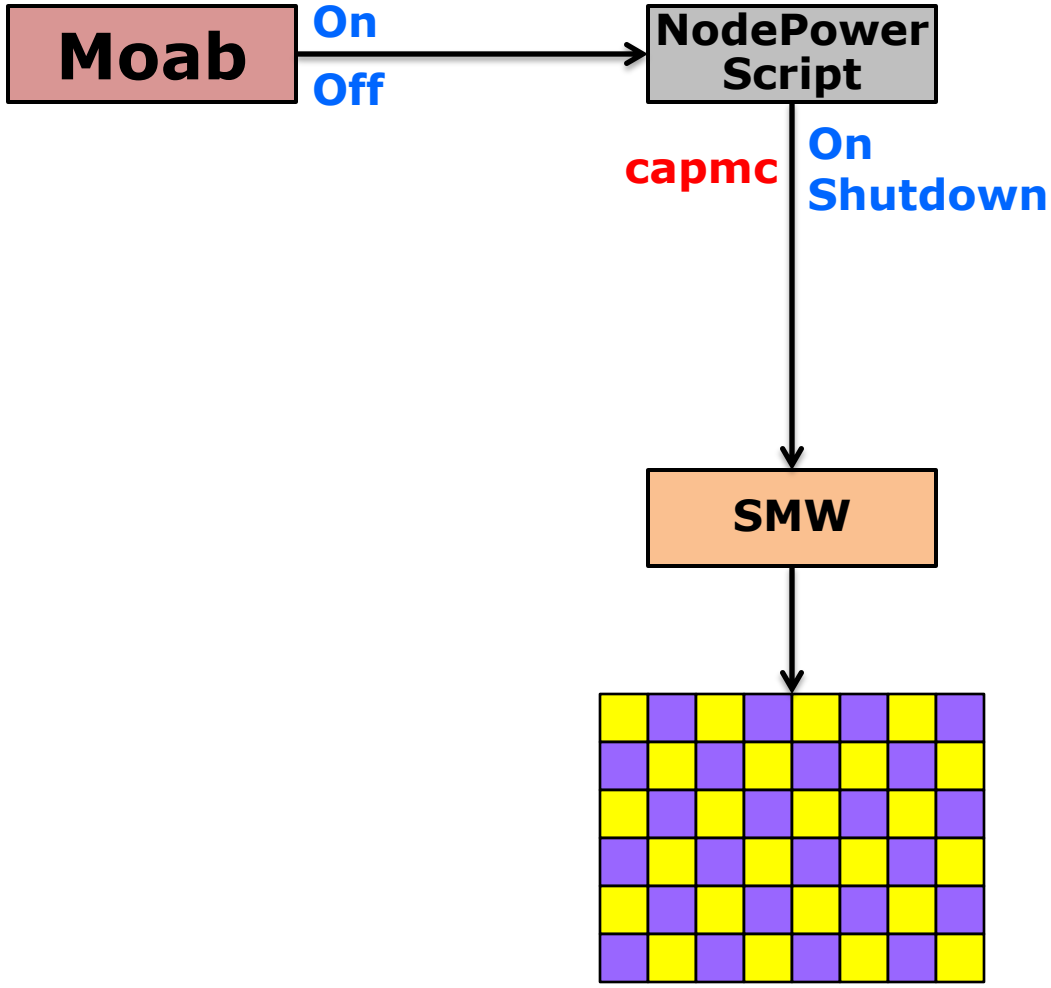
Node Power State Management

- **Additional Power States (C-states)**
  - Power management for idle nodes
  - Suspend (C3/C6)
  - Hibernate (S4)
  - Shutdown (S3)

Power Management Control Architecture



Moab/Cray Power Management Control Architecture



Per-Job Energy Accounting

- **Part of TORQUE/Cray integration**
- **End of job**
  - TORQUE checks for RUR-based energy consumption information
  - Sums energy use for each job step (aprun)
  - Sends energy_used job metric to Moab
- **Moab passes job metric to Moab Accounting Manager for possible charging**

DataWarp Integration

DataWarp Integration Project

- **Supports all DataWarp storage types**
  - Per-Job DW storage
  - Persistent DW storage (across multiple jobs)
- **All DW storage scheduled by Moab**
- **Out-of-the-box functionality**
  - Some site-specific information configuration required

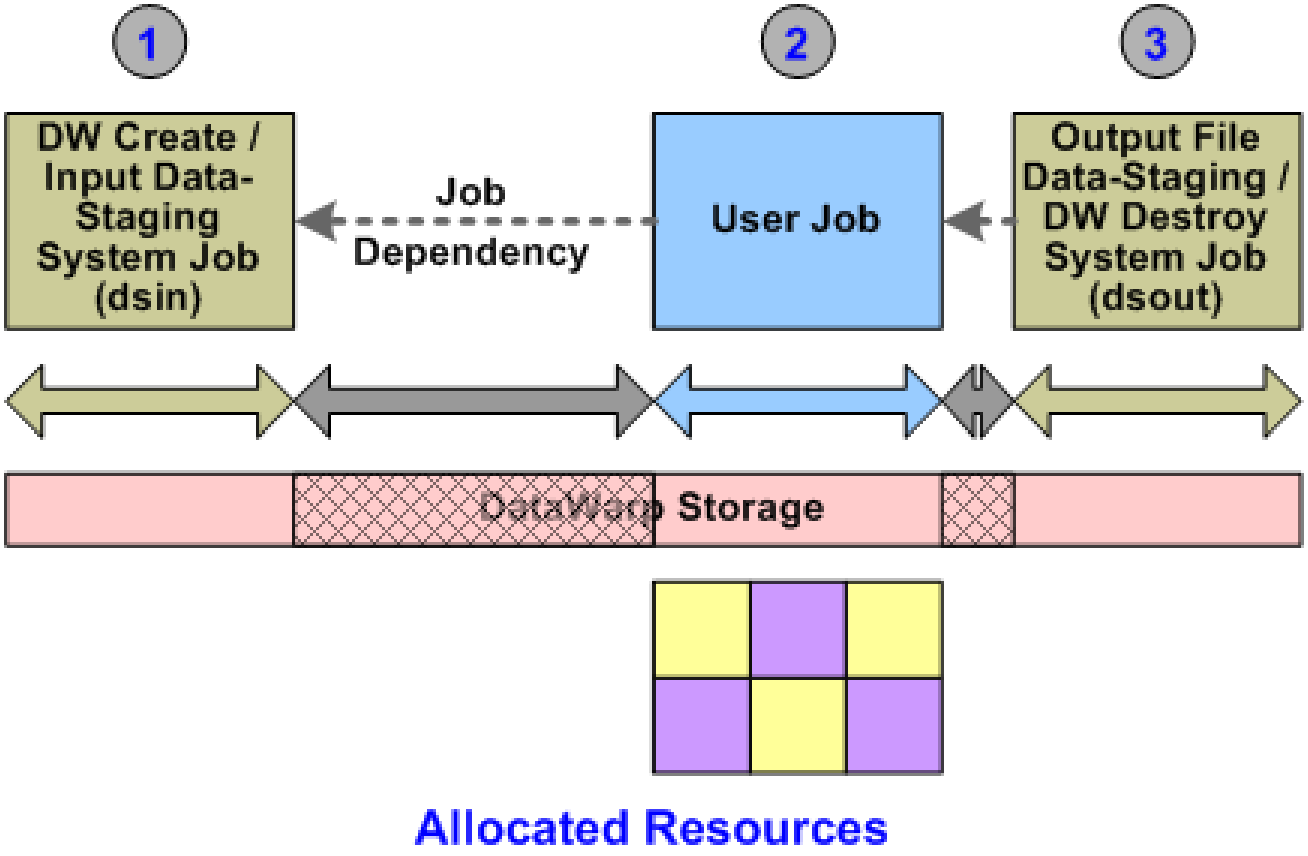
DataWarp Integration

- **2 per-job DW storage models supported**
  - Selectable by user at job submission
- **3-job Model**
  - Favors compute nodes as “scarce” resource
  - Compute nodes allocated only during user job
- **Prologue/Epilogue Model**
  - Favors DW storage as “scarce” resource
  - Compute nodes allocated during entire DW storage allocation

3-Job Model

- **Follows Moab Data-staging Model**
  - Uses Moab "system" jobs
- **Automatically creates 3-job workflow**
  - DW creation/input data staging
  - User job
  - Output data staging/DW destruction

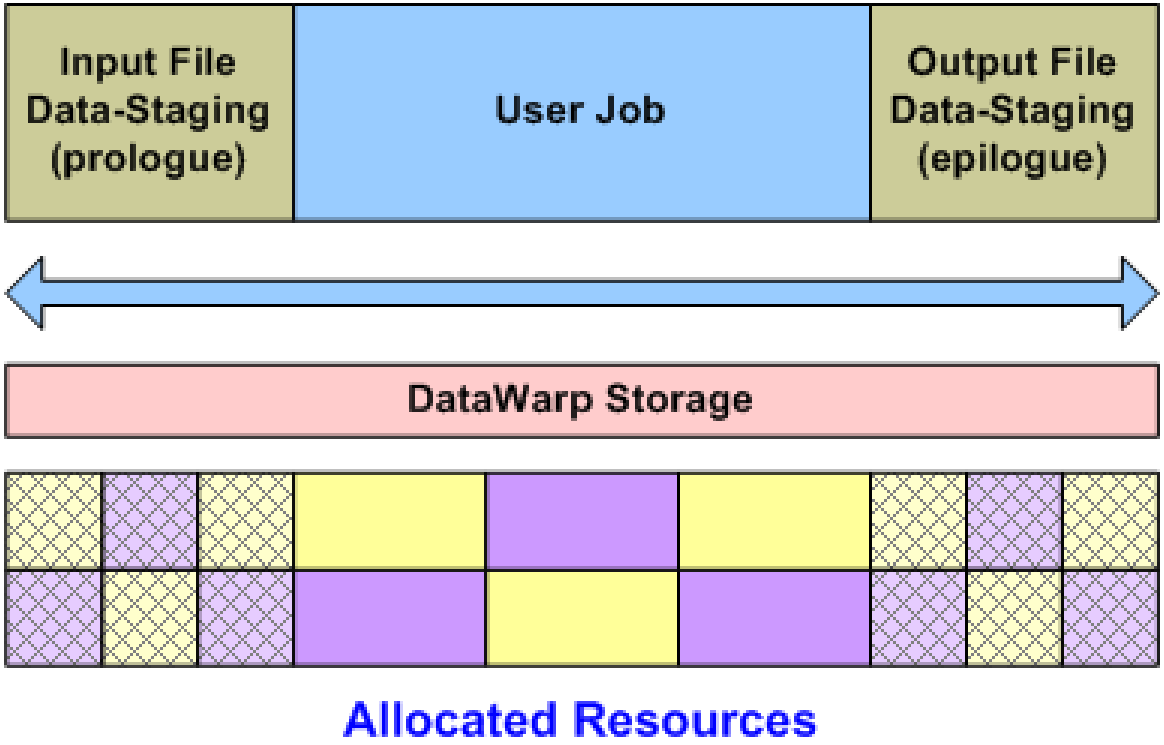
3-Job Model Architecture



Prologue/Epilogue Model

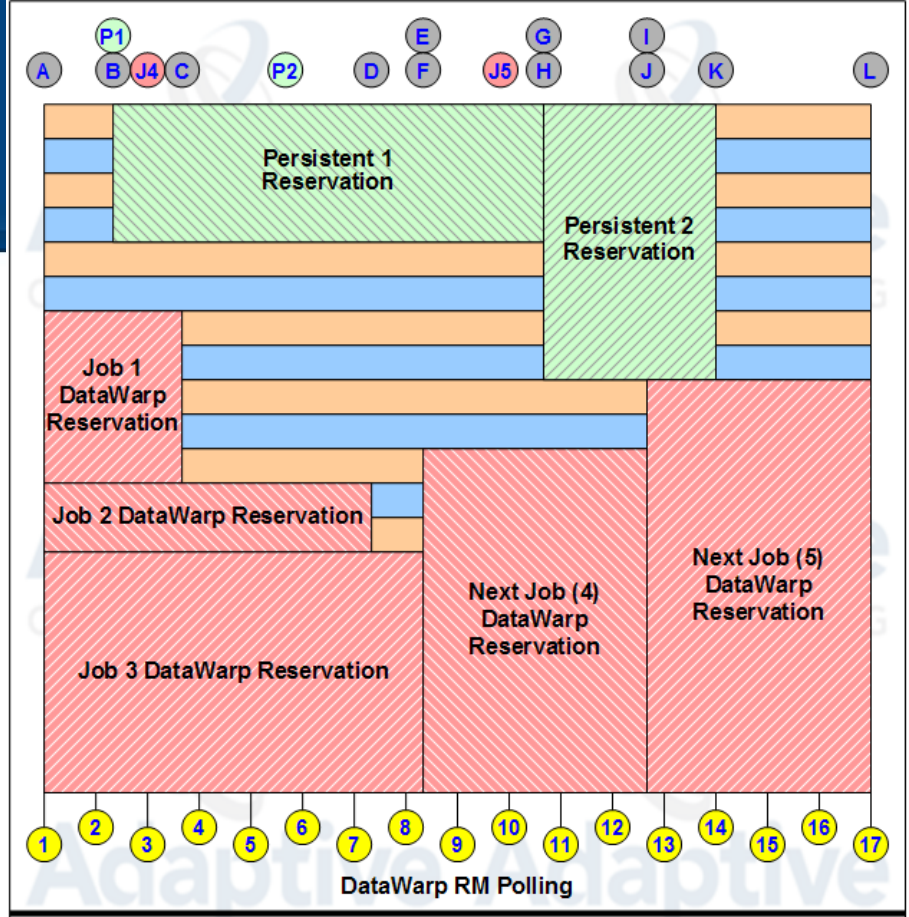
- **Uses TORQUE prologue and epilogue**
- **Prologue**
  - Create job's DW storage allocation
  - Stage input data to DW storage
- **User job**
- **Epilogue**
  - Stage output data from DW storage
  - Destroy job's DW storage allocation

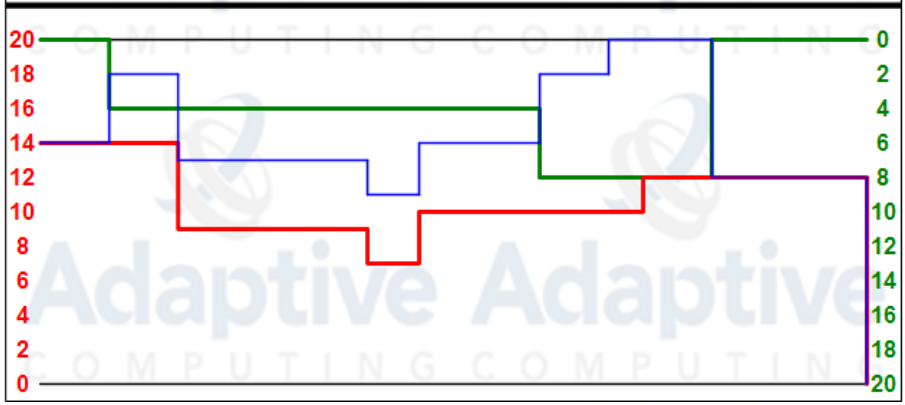
Prologue/Epilogue Model Architecture



Persistent DataWarp Storage

- **Moab tracks DW storage capacity**
- **Moab persistent DW storage reservation**
- **Persistent DataWarp storage management**
  - Created by reservation start trigger
  - Destroyed by reservation end trigger
- **Jobs get mount point**





Adaptive COMPUTING

Nitro
HTC Job Scheduler

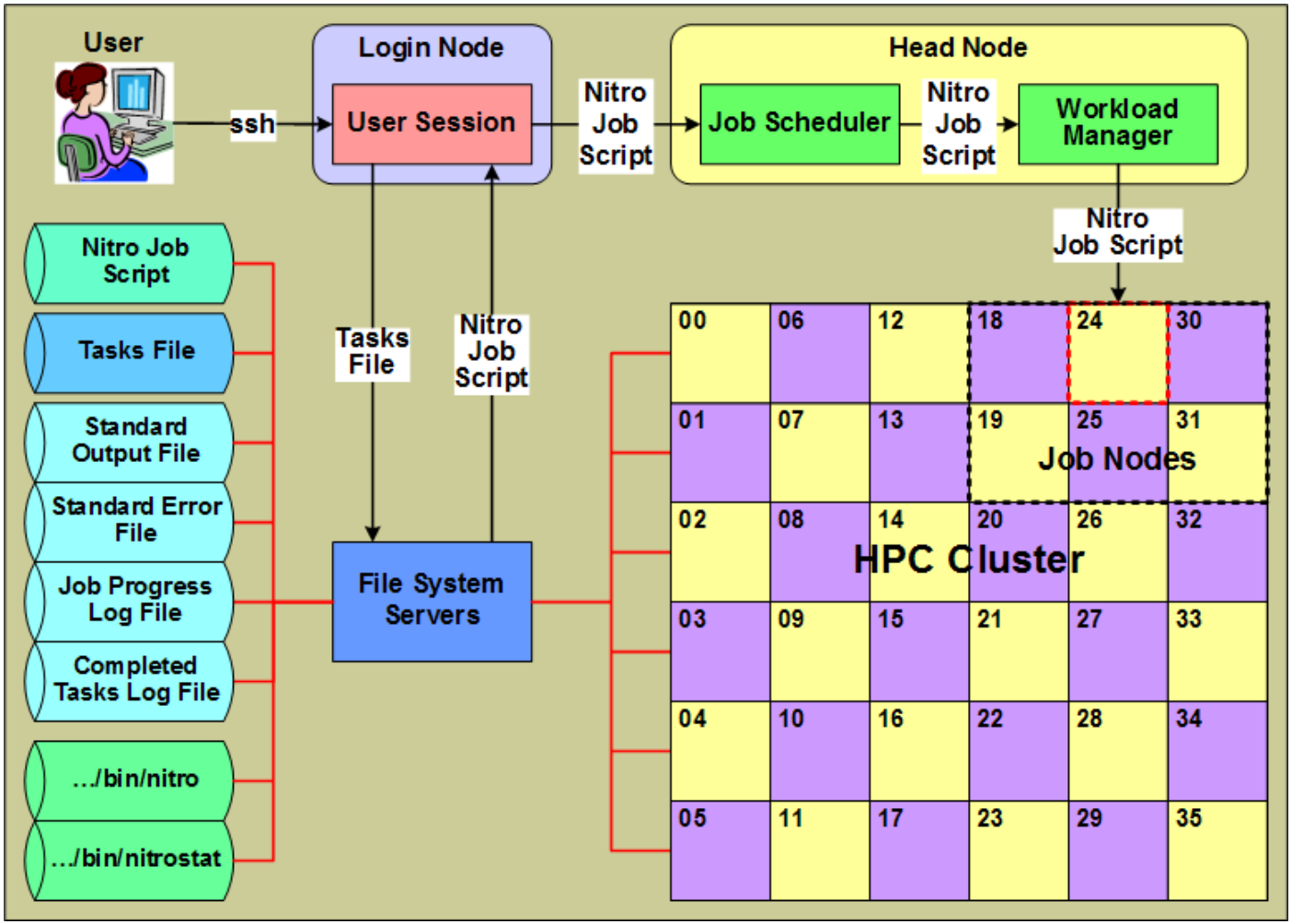
Nitro Product

- **High-throughput Computing (HTC)**
  - Usually short, single-core jobs
- **Nitro HTC Scheduler Product**
  - Eliminates traditional scheduler overhead for HTC jobs
  - Executes HTC jobs very fast
    - Shorter the HTC jobs, the greater the speedup
  - Submitted as normal batch job to HPC job scheduler
  - Scheduler-agnostic

Nitro Architecture

- **Nitro Coordinator**
  - Executes on first host allocated to Nitro job
  - Schedules all HTC tasks defined in “Tasks” file
  - Assigns tasks to Nitro “workers” for execution
- **Nitro Workers**
  - Execute on all hosts allocated to Nitro job
  - Worker executes assigned tasks as fast as possible
  - Can run more tasks than “cores” on worker host
- **Task File**
- **Log Files**

Nitro Product Architecture



Nitro Job Progress Log

- **In-progress and Final versions**
- **Nitro Job Information**
  - Start / Finish/ Elapsed Times
  - Job ID
  - Task / Job Progress Log / Completed Task Log File Paths
  - Success/Failure/Timeout/Invalid Task Statistics
  - Average Task Time
  - Tasks per Second
  - Worker Statistics

Completed Task Log

- **All tasks**
- **Task Status**
  - Success
  - Failure
  - Timeout
  - Invalid task definition
- **Tab-delimited format**
  - Spreadsheet, database, etc
- **Optional column headings**

Task Information

- Job ID
- Task ID
- Line Number
- Task Status
- Exit Code
- Host Name
- Task Start Time
- Task Duration
- User CPU Time
- System CPU Time
- Core Count
- Max Physical Memory
- Max Virtual Memory
- Task Name (optional)
- Labels (optional)
- Failure Message (optional)

Nitro "Linger" Mode

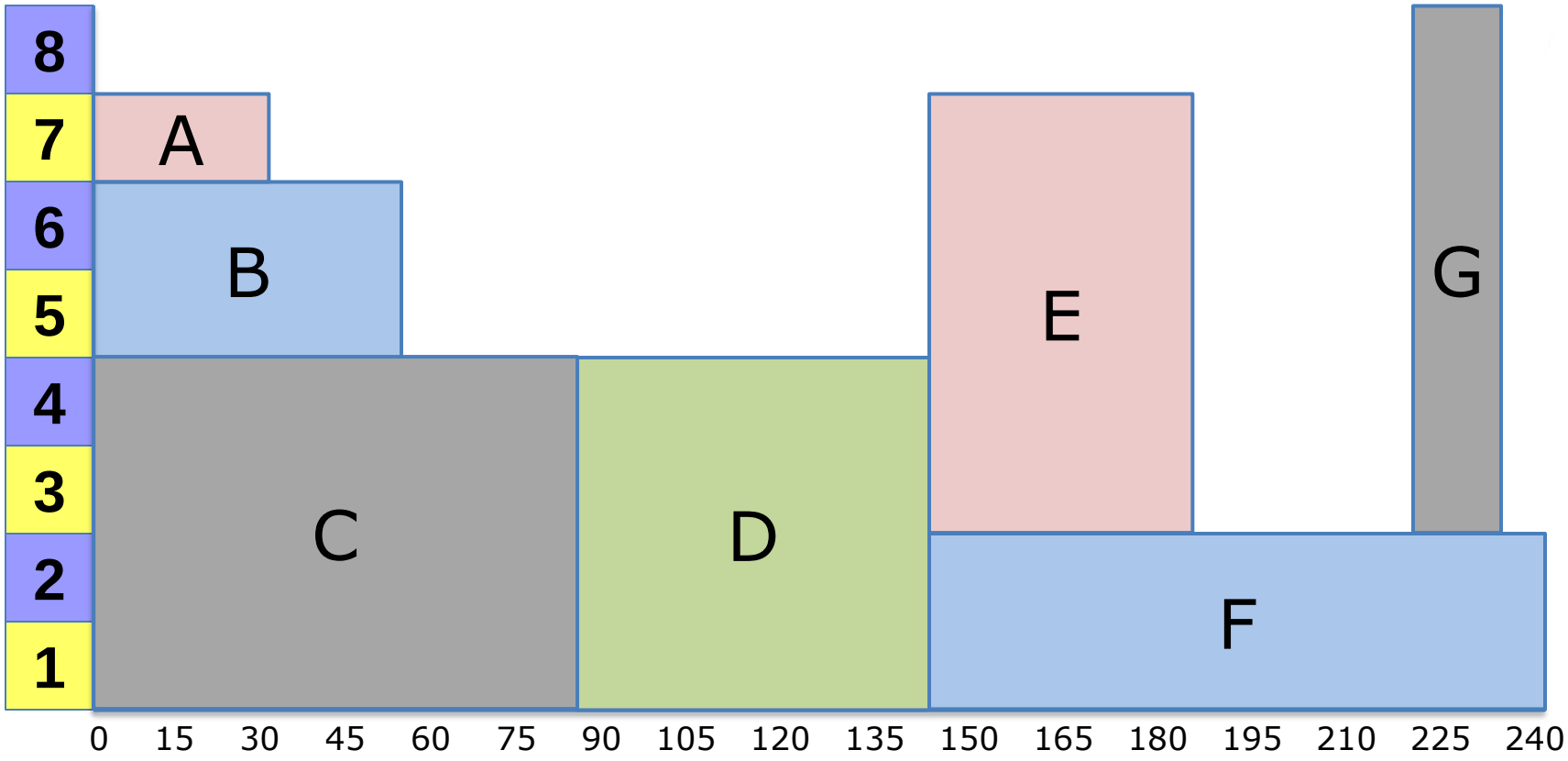
- **--linger Command Line Option**
- **Coordinator continues execution after Task file end-of-file encountered**
  - Periodically polls Task file
- **User appends new task definitions to Task file**
  - Nitro executes additional tasks
- **Statistics → Last 60 seconds**

Collaboration Invitation: Malleable/Evolving Job API

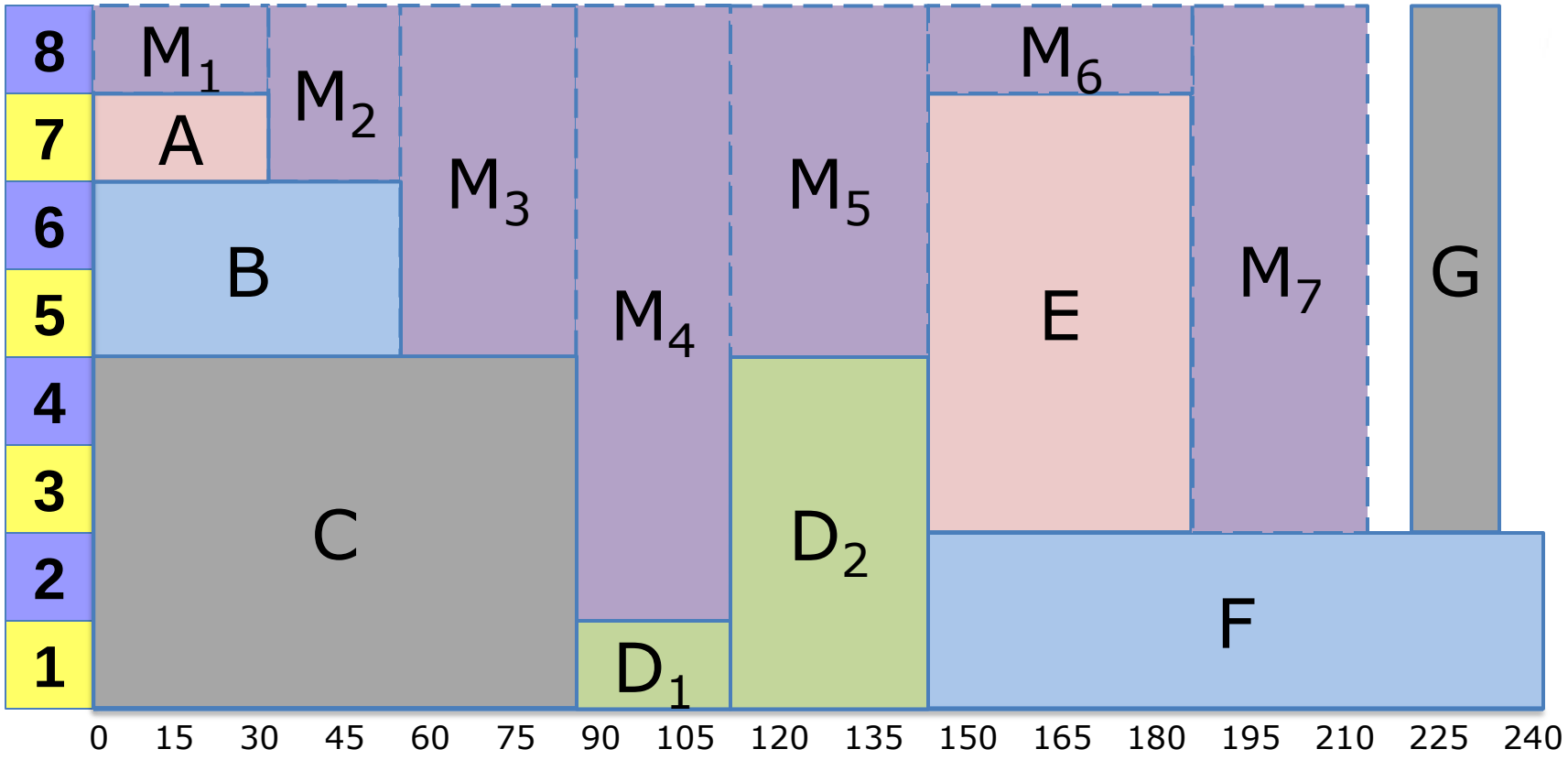
Job Types and Scheduling

Classification	Description
Rigid	A job that requires a fixed set of resources, all of which the batch system must allocate to the job before it starts *(fixed static resource allocation)*
Moldable	A job that allows a variable set of resources for scheduling but requires a fixed set of resources to execute, which the batch system must allocate before it starts the job and which the job must discover in order to execute properly *(variable static resource allocation)*
Malleable	A job that allows a variable set of resources, which the batch system dynamically allocates and deallocates and of which allocation/deallocation the scheduler must inform the running job so it can adapt to the new resource allocation *(unidirectional, scheduler-initiated and application-executed, variable dynamic resource allocation)*
Evolving	A job that dynamically requests and relinquishes resources during its runtime and of which the scheduler must be informed while the job is running so it can allocate or deallocate the resources *(unidirectional, application-initiated and scheduler-executed, variable dynamic resource allocation)*
Dynamic	*Adaptive Computing Definition* A job where the scheduler or the job can dynamically initiate resource allocation changes during the job's runtime and of which the other must be informed while the job is running so they both keep the job's resources allocation synchronized *(bidirectional, application- or scheduler-initiated and scheduler- or application-executed, variable dynamic resource allocation)*

System Utilization with No Malleable or Evolving Jobs



System Utilization with Malleable Job M and Evolving Job D



Standard Malleable/Evolving Job API

- **Currently no standard application API**
- **Much research done**
  - Univ of Illinois at Urbana-Champaign (UIUC)
    - Malleable jobs (Charm++ codes)
  - German Research School for Simulation Sciences (RWTH Aachen) (group now at Technische Universität Darmstadt)
    - Evolving jobs (AMR code with Maui/TORQUE)
- **Invitation to start API collaboration**
  - Application Developers
  - Scheduler Vendors
- **Email: gbrown@adaptivecomputing.com**

Questions?