Scheduling Diverse High Performance Computing Systems with the Goal of Maximizing Utilization

Troy Baer
Senior HPC System Administrator
National Institute for Computational Sciences
University of Tennessee

National Institute for Computational Sciences

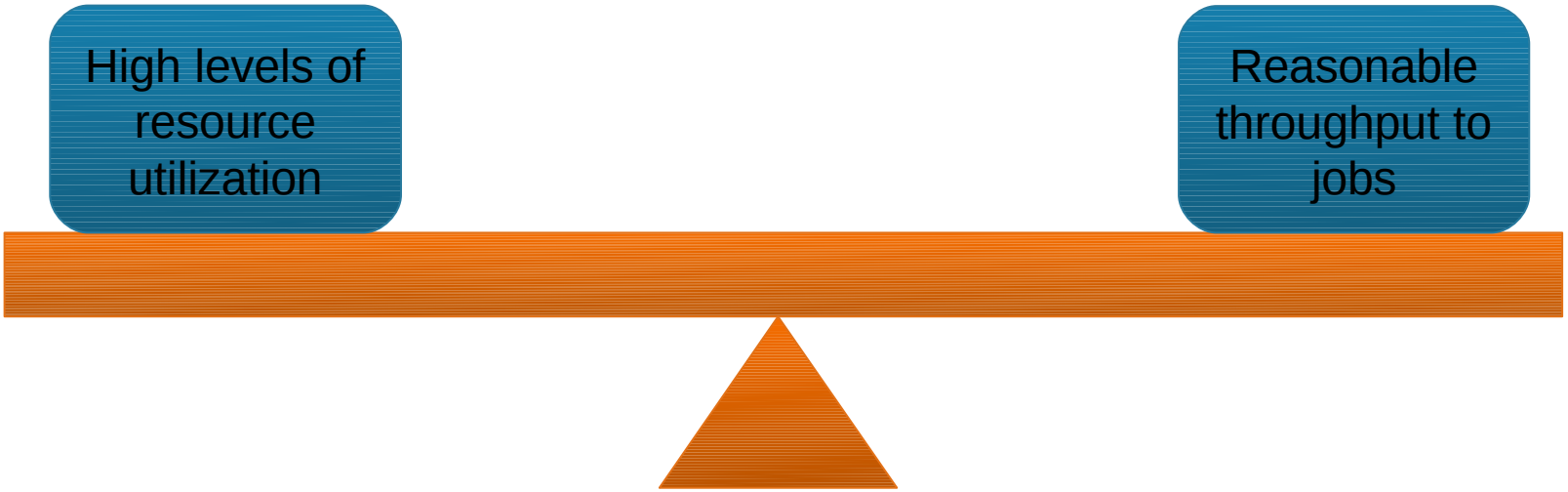
University of Tennessee & ORNL partnership

- **NICS is a NSF HPC center**
  - **Operated by UT, located at ORNL**
  - **XSEDE (formerly Teragrid) Service Provider**
- **Current Systems**
  - **Kraken (Cray XT5, 112,984 cores)**
  - **Nautilus (SGI UV, 1,024 cores)**
  - **Keeneland initial delivery system (HP GPU cluster, 1,440 cores + 360 GPUs) in conjunction with Georgia Tech**

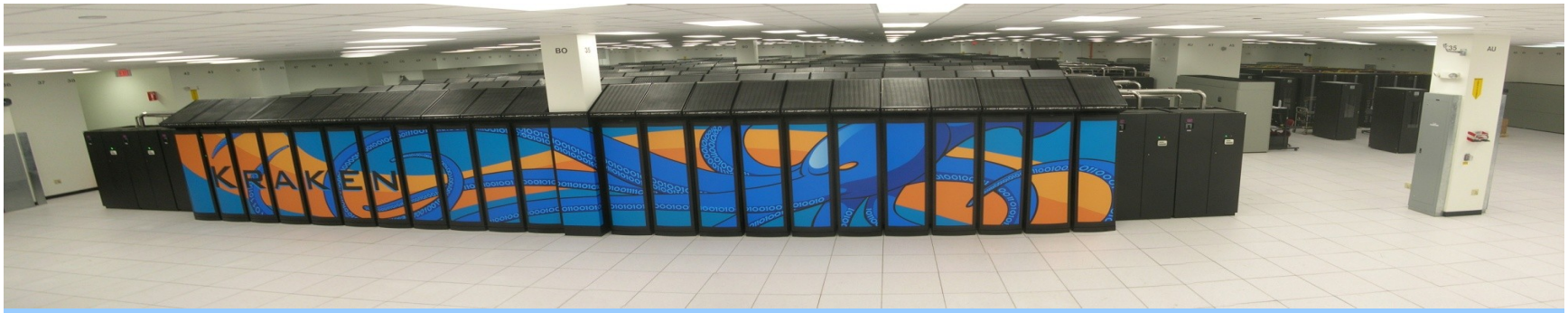
Overview

- **Introduction to NICS**
- **Scheduling on Kraken**
  - **Scheduling Overview**
  - **Need for bimodal scheduling**
  - **Capability Scheduling using Fence Reservations**
  - **Capability Scheduling using Triggers**
- **Scheduling on Keeneland**
  - **Scheduling Overview**
- **Future Work**

Goal of a HPC Service Provider

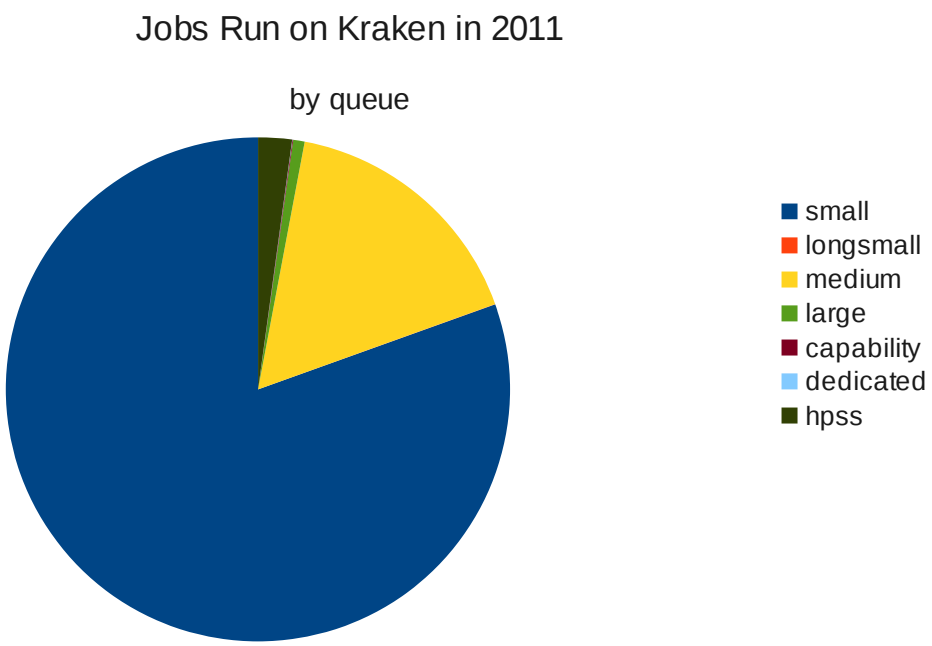


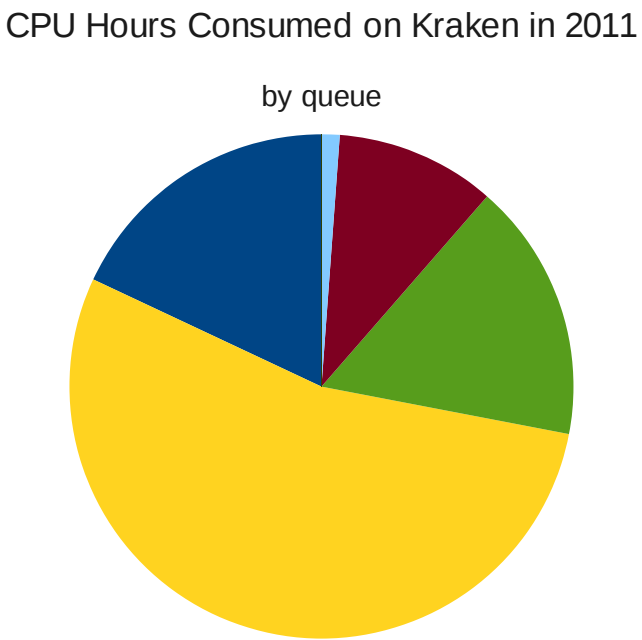
Kraken XT5 Specifications



Compute processor type	AMD 2.6 GHz Istanbul-6
Compute cores	112,896
Compute sockets	18,816 six-core
Compute nodes	9,408
Memory per node	16 GB
Total memory	147 TB
Peak system performance	1.17 PF
Interconnect topology	22 x 16 x 24 Torus/Seastar2+
Parallel file system space	3.3 PB (raw) 2.4 PB (usable)
Parallel file system peak performance	30 GB/s

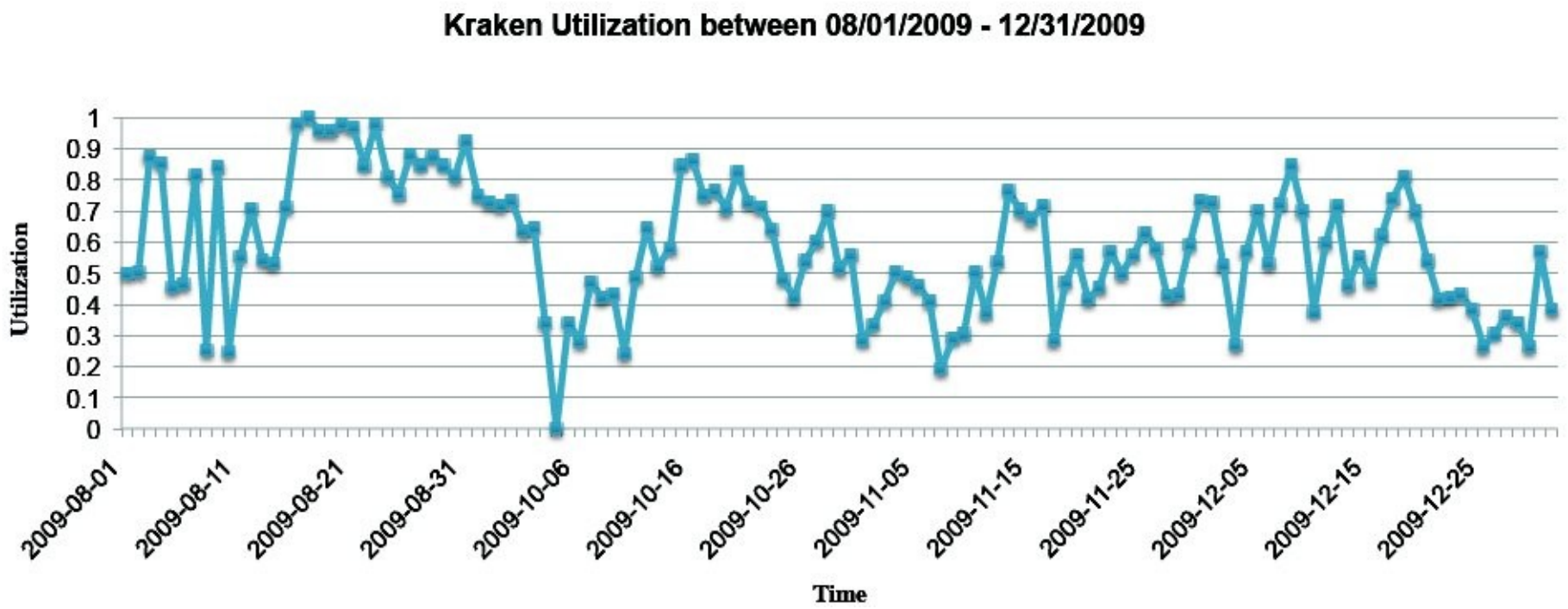
Job Mix on Kraken During 2011





Need for Bimodal Scheduling

- **Prior to January 2010 scheduling was on a strictly priority basis**
- **Peak and valley problem in utilization**



Need for Bimodal Scheduling

- **Capability users had to wait a long time before their jobs could run**
- **Capability jobs often did not manage I/O properly, leading to frequent Lustre problems**
- **This led to a need for a stable and more predictable scheduling system**

Initial attempt at Bimodal Scheduling

- **Friendly user period after Istanbul upgrade on Kraken**
  - **Largely manual scheduling – inspecting the queued jobs and adjusting priorities or running jobs manually**
  - **Resulted in ~20% increase in utilization**
- **Efforts to reduce the amount of manual intervention required**
- **Removal of expansion factor priority component**
  - **Caused priorities to change in ways that were difficult to explain to users**
  - **Remaining priority components were number of cores request and queue wait time**

Problems Running Dedicated jobs

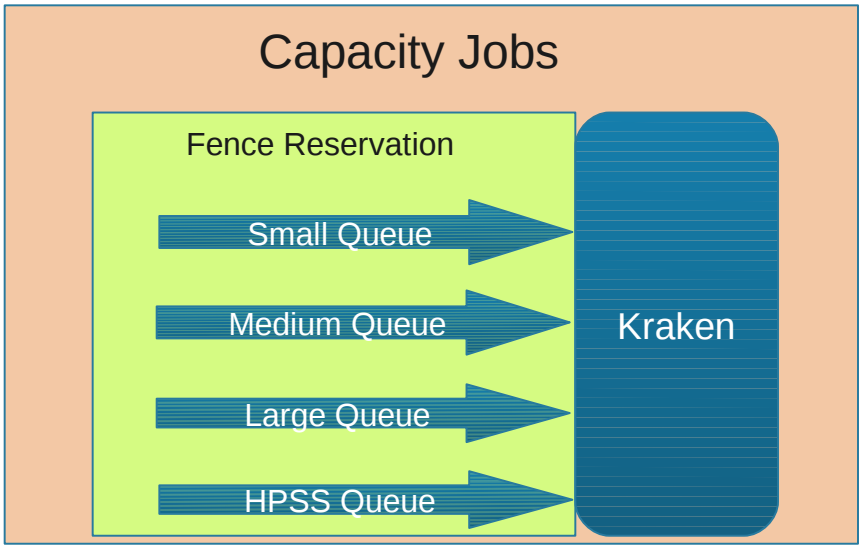
- **Initially run manually with scheduling paused**
- **Disadvantage of this approach:**
  - **On call sys admin had to start next dedicated job (Runs usually lasted 24 – 48 hours)**
  - **Every person involved with job had to stay up to monitor progress**
  - **Backfill jobs had to be run manually as well**

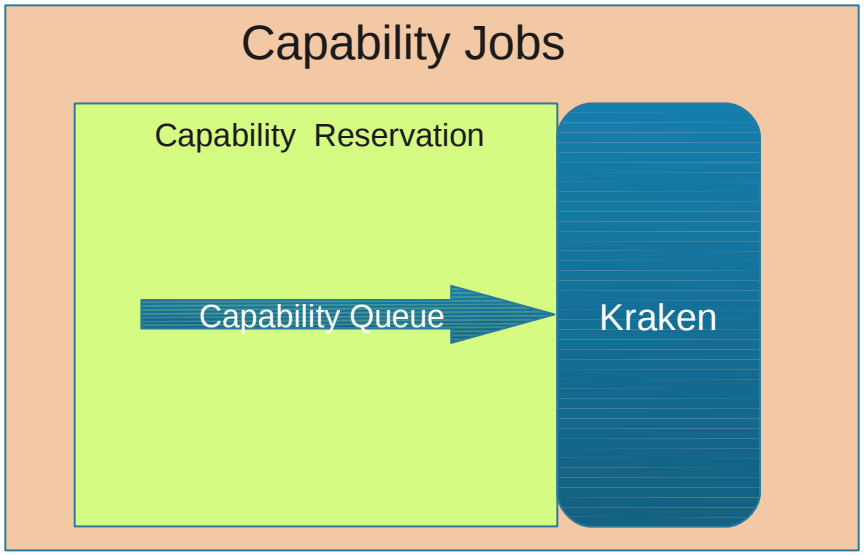
The Moab/ALPS disconnect

- **Moab/ALPS disconnect**
  - **ALPS node health checker runs after every job**
  - **Moab assumed nodes were ready as soon as job was ended, ignoring ALPS health checker**
  - **ALPS reported the nodes were not available immediately**
  - **Jobs would be requeued after failing to acquire nodes until health checker completed**
  - **In some cases, this led to non-capability jobs running while there were capability jobs left in queue**
- **Human errors also led to premature termination of capability jobs**
- **By end of 2010, through trial and error, an acceptable level of reliability was achieved but process was still heavily manual**

Capability Scheduling Using Fence Reservations

- **Fence reservations: Standing reservations set up on Kraken with a granularity of 1 day**
- **Reservation controlled by an ACL which specified which queues could run on the machine**



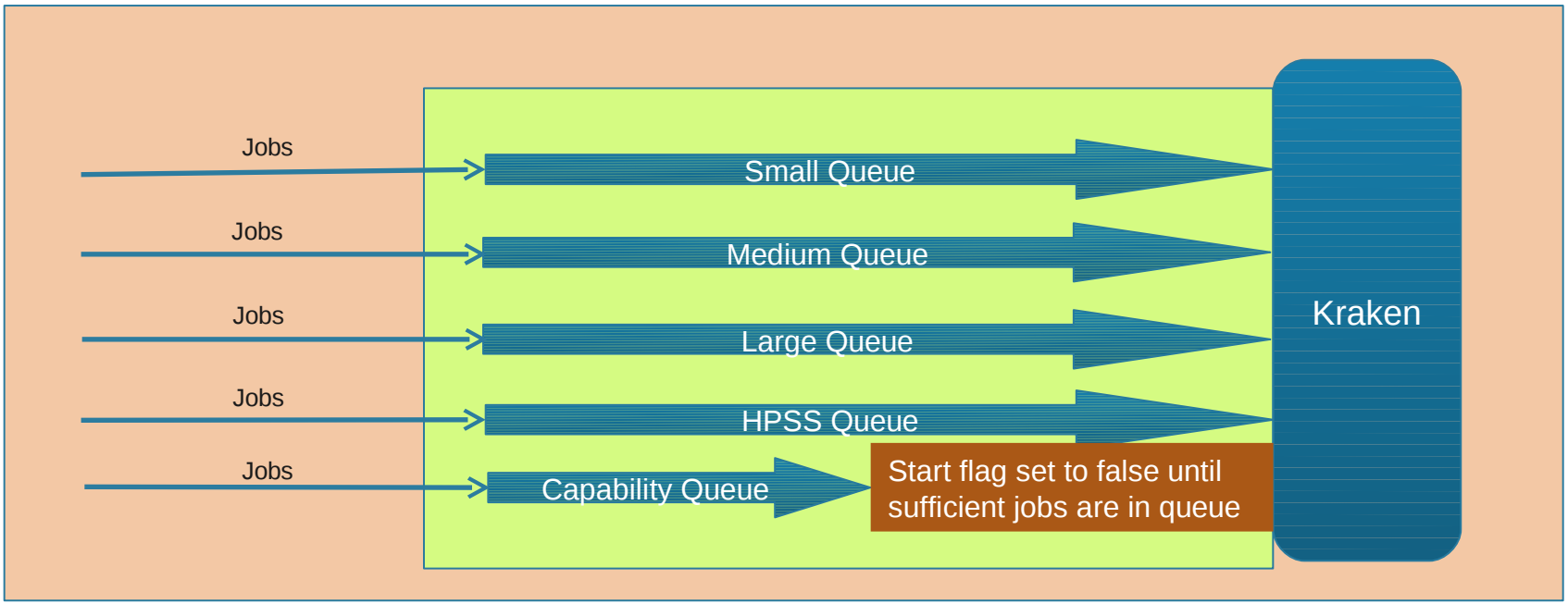


Drawbacks of Capability Scheduling Using Fence Reservations

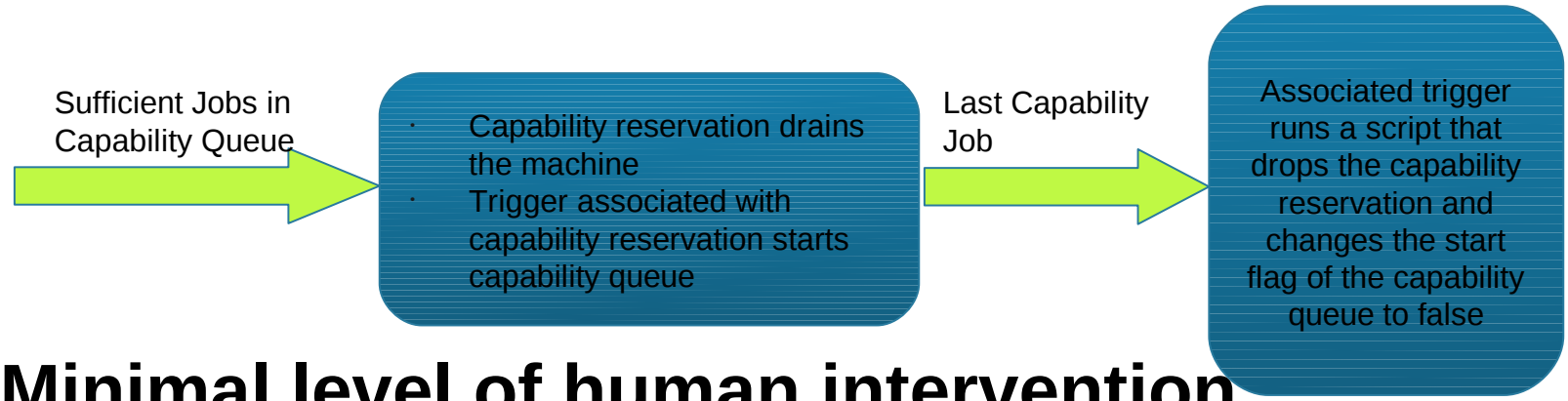
- **Fence reservation had to replace capability reservation as soon as last capability job was done, else machine would idle**
- **Too many reservations in flight**

Capability Scheduling Using Triggers

- **Moab version 5.4 changed the way reservations were handled**
- **From version 5.2, Moab could launch triggers**
- **Triggers enable actions when certain events, offset or threshold criteria are satisfied**



Running Capability Jobs Using Triggers

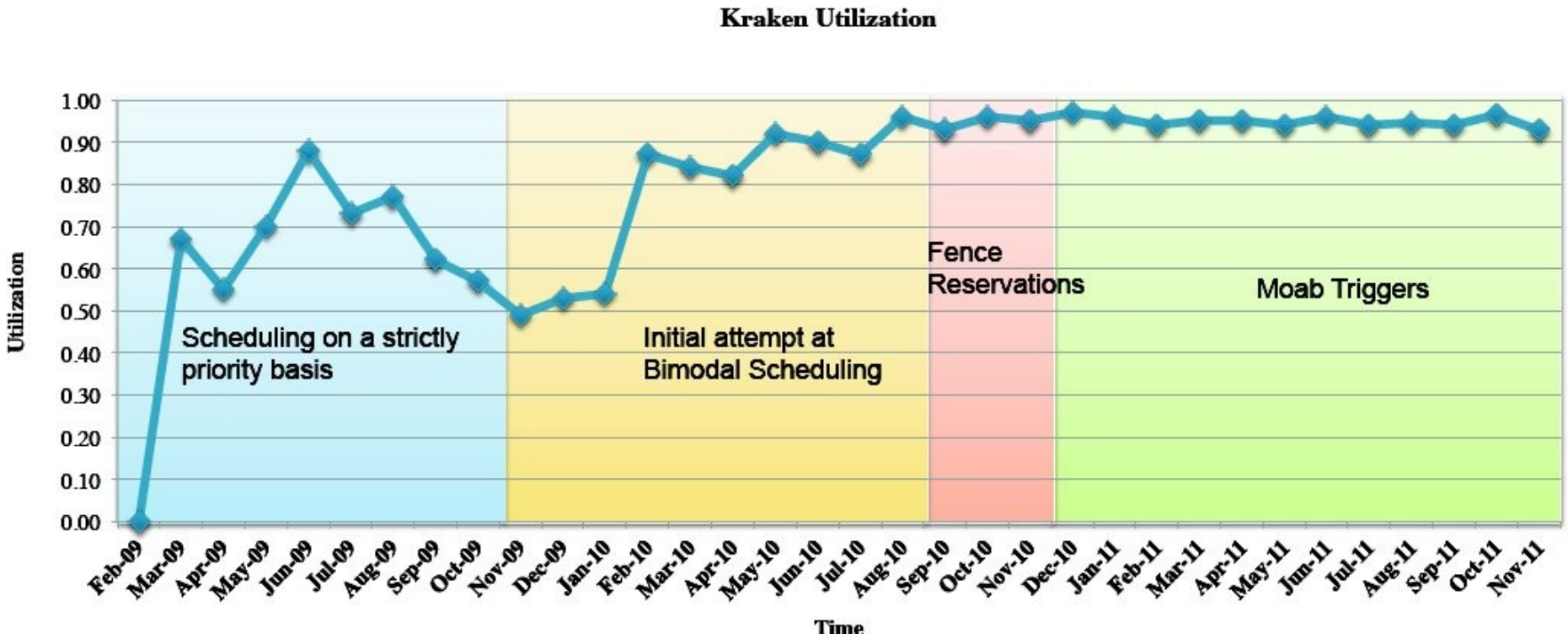


- **Minimal level of human intervention**
  - **Reservation profile used to simplify reservation creation**
  - **Capability queue off by default, turned on and off by triggers**
- **Allows capability users to debug and resubmit jobs within the capability window**
- **Turnaround time for capacity users unaffected**

Current Scheduling on Kraken

- **Capacity Jobs – use <= 49,536 cores**
  - **Run most of the time**
- **Capability Jobs – use between 49,537-98,352 cores**
  - **Run after maintenance windows or crashes**
- **Dedicated Jobs – use > 98,352 cores**
  - **Extremely I/O intensive**
  - **Scheduled roughly once every quarter**
  - **Dedicated jobs with wallclock requests <= 1 hour may run just after a PM or crash**

Overall Utilization on Kraken



Scheduling on Keeneland

Keeneland - Initial Delivery (KID)

CPU	2 x Intel Xeon X5660 2.80 Ghz (Westmere)
GPU	3 x Nvidia Tesla M2070 6GB (Fermi)
Interconnect	Infiniband QDR (single rail)
# Compute Nodes	120
Total CPU cores	1,440
Total GPUs / cores	360 / 161,280
# Login Nodes	4
# Management Nodes	2

Scheduling on Keeneland

- **Keeneland is currently a development system open to friendly users**
- **Fairshare Policy:**
  - **Each project have a default fairshare target of 10%**
  - **To prevent users from “hogging” the machine:**
    - **Maximum eligible jobs per user (at a time): 5**
    - **Maximum eligible jobs per project (at a time): 10**
- **Capability reservations using triggers similar to what's done on Kraken**
  - **Bug in Moab 6.1.x where reservations created using a profile don't get the profile's CLASSLIST ACL – have to add by hand**
  - **This problem now affects Kraken as well after CLE 3.1 upgrade**

Future Work

- **On Kraken, no way currently to distinguish between jobs submitted to capability queue at the start of the reservation and jobs submitted after the reservation was open**
  - **Problem of determining last job to attach trigger to**
  - **Attach higher priority to jobs submitted before capability queue was open**
  - **Alternatively, hold capability jobs by default and release eligible jobs at the start of the capability window**
- **Freeing up cores as capability run progresses**

Questions

Appendix A: Anatomy of a Fence Reservation

```
SRCFG[fence] COMMENT='fence in capability jobs to
run only on certain days'

SRCFG[fence] PARTITION=xt5 TASKCOUNT=8256

SRCFG[fence] PERIOD=DAY DEPTH=7
DAYS=Mon,Tue,Thu,Fri,Sat,Sun

SRCFG[fence] STARTTIME=00:00:01 ENDTIME=23:59:59

SRCFG[fence]
CLASSLIST=small,longsmall,medium,large,dmover,hpss

SRCFG[fence] FLAGS=SPACEFLEX
```

Appendix B: Anatomy of a Trigger-Based Capability Reservation

```
RSVPROFILE[capability]
TRIGGER=AType=exec,EType=start,Action="/var/control_capability
start $OID"

RSVPROFILE[capability]
TRIGGER=AType=exec,EType=end,Action="/var/control_capability
end $OID"

RSVPROFILE[capability]
TRIGGER=AType=exec,EType=cancel,Action="/var/control_capability
cancel $OID"

RSVPROFILE[capability] CLASSLIST=capability,dedicated

RSVPROFILE[capability] FLAGS=DEDICATEDRESOURCE
```