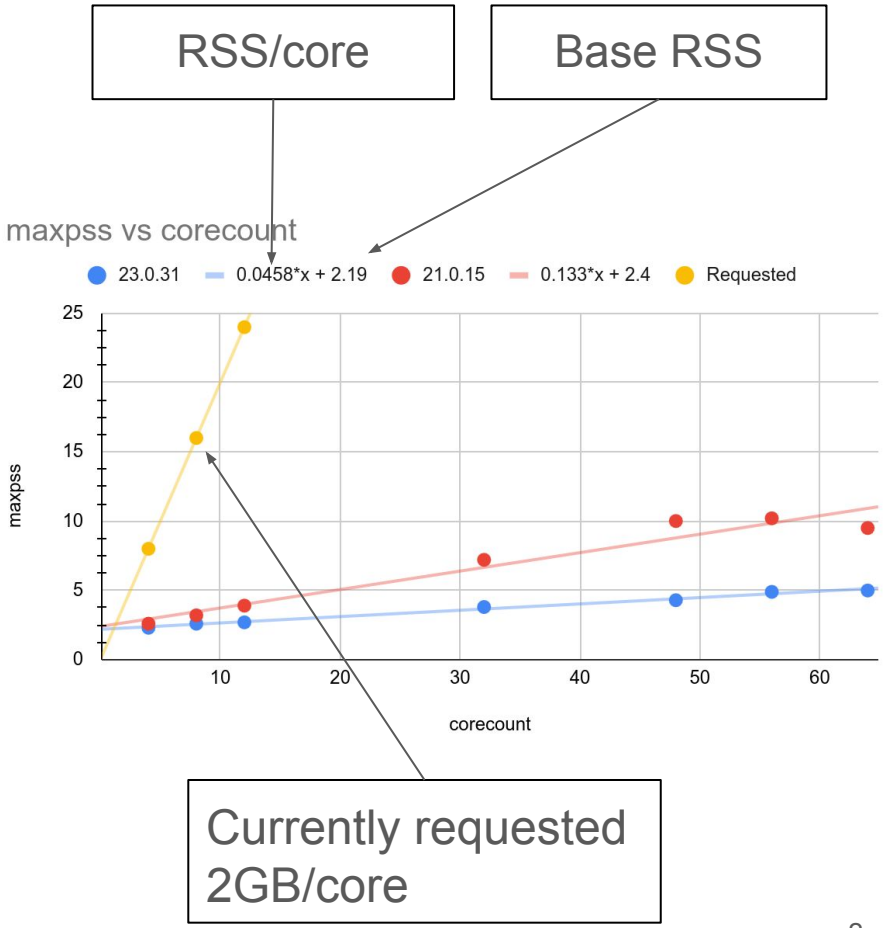
Flexible memory usage for ATLAS

- Rod Walker , LMU

Motivation

- ATLAS have high memory workloads
  - some irreducible e.g.Sherpa evgen, HI, AOD merge, …
    - Reasons: HepMC3, AF3(fastsim), #volumes.
  - millions of histograms for user systematics, ML
- Grid hardware does not change quickly
  - **not** asking to buy more RAM per core
  - need to make better use of what we have.
- MCORE simulation uses very little RSS
  - 300MB/core on 8 cores, 170MB/core on 16
  - many other workloads <1GB/core
  - all currently reserve 2GB/core
- Goals
  - more cores for high RSS workloads
  - at more sites for colocation with data
    - reduce need to transfer input data

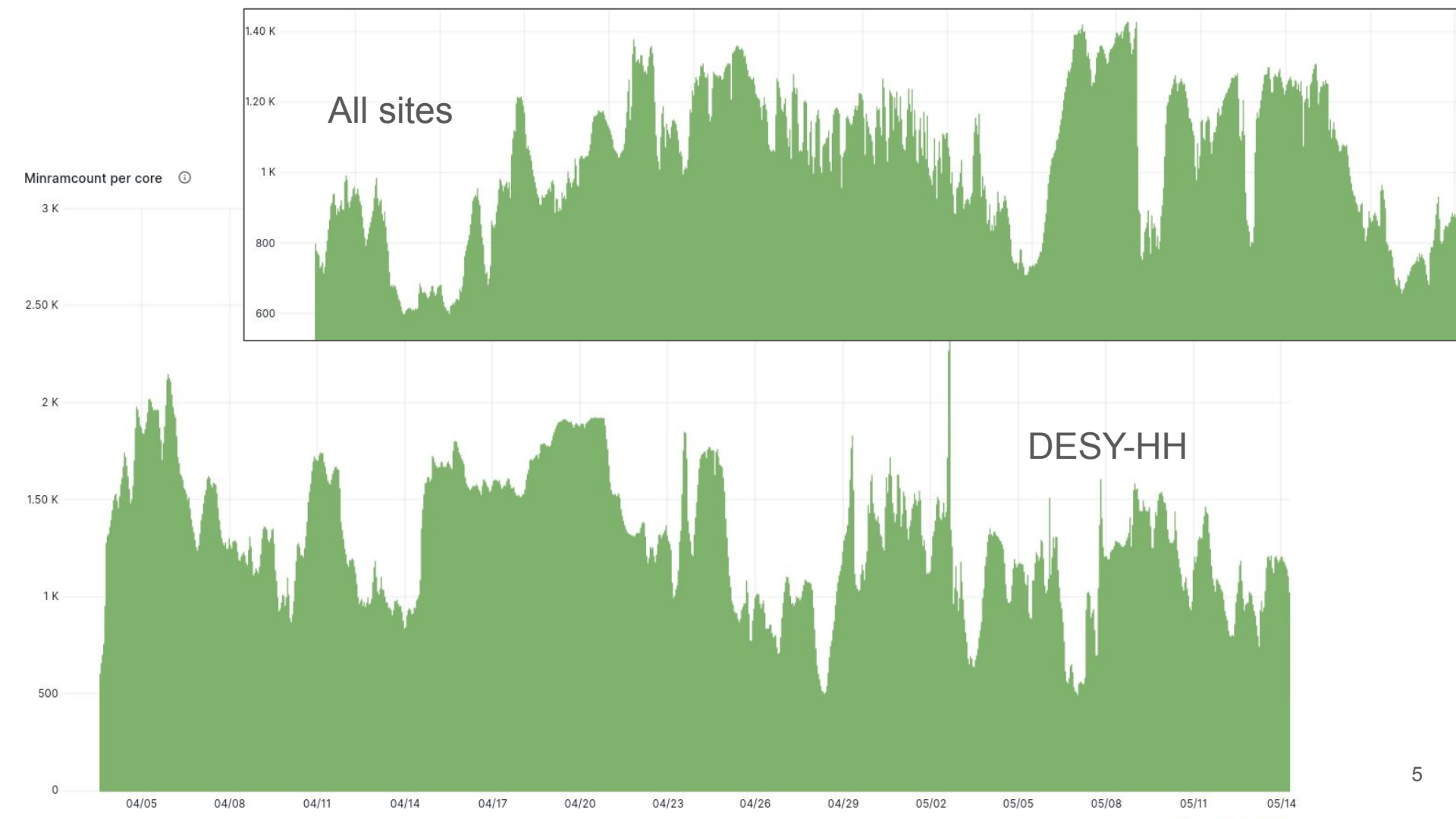


How to run himem at more sites

- Run mix of high and low memory jobs
  - keeping mean requested memory/core below the physical value
- Submit pilot with requirements that CE passes to BS
  - Batch Systems can pack nodes according to requirements
  - mix hi and lomem jobs on a *node* to keep all cores full
- *Pull* model has streams of pilots with the same requirements
  - currently 2 memory ranges(per core): 0-2GB, 2GB-maxrss
  - dev ongoing to increase granularity and include very lomem, e.g. 0-1,1-2,2-4,4-6
- *Push* pilot submitted with specific requirements of pre-loaded job
  - MB granularity on memory, works today and in use
- Good news: nothing for sites to do
  - CEs and BSs support this already

Maintaining job mix

- Staying below 2GB/core on site avoids admin grief and accounting issue
  - 2GB is site dependent, often higher.
- Have crude limit to stop himem jobs
  - resource_type_limits.HIMEM - limit # running cores
- better mechanism to stay below site meanrss/core (in dev)
  - running job sum(job.ramcount)/sum(job.corecount) > site.meanrss GB/core
    - stop dispatch of jobs with ramcount>site.meanrss
  - overshoot and oscillation may need tweaks
- What if we want to fill resources with himem?
  - leaves cores idle so needs to be accounted for
  - propose to dodge this, for now, by maintaining job mix (or see backup slide)


All sites
1.40 K
1.20 K
1 K
800
600
Minramcount per core
3 K
2.50 K
2 K
1.50 K
1 K
500
0
DESY-HH
04/05
04/08
04/11
04/14
04/17
04/20
04/23
04/26
04/29
05/02
05/05
05/08
05/11
05/14

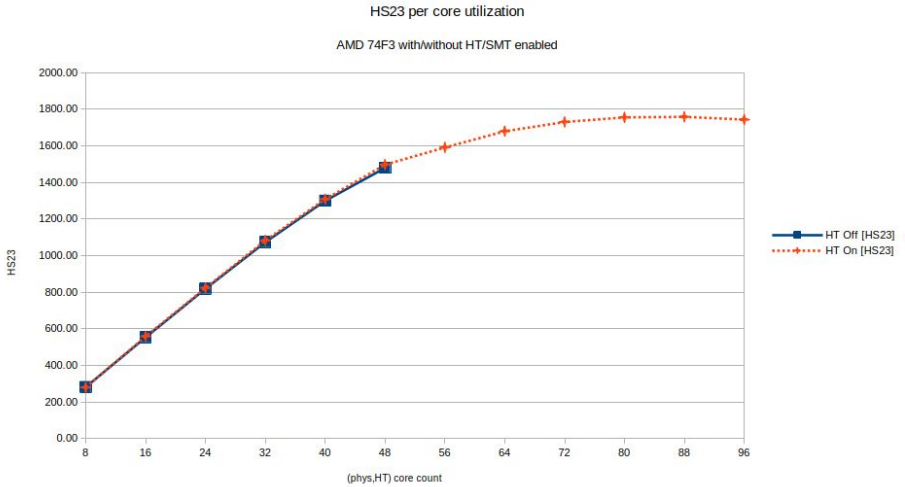
16 core standard slots

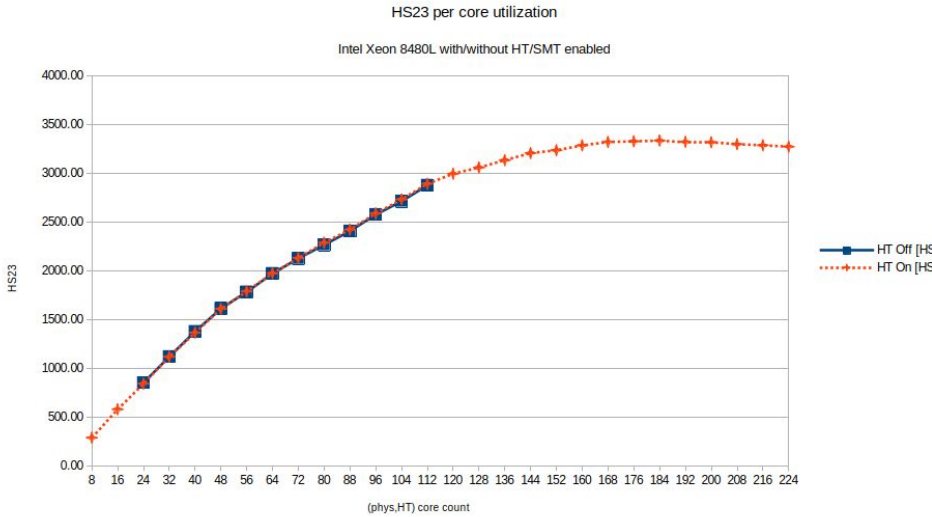
2015	Broadwell **16** cores
2023	EPYC **64**, Sapphire Rapid **56**

- CPU cores per node more than doubled since 8 was chosen
  - Multicore TF summary in CHEP 2015.
- Easy win on scalability of job & output file handling
  - implies doubling the size of jobs, e.g. #events
  - ~halves RSS/core for MT sim(300->170MB/core)
- Multi-threaded and multi-process payloads have good efficiency @ 16cores
  - serial phases << parallel phase, no worse than 2*8 core jobs
- Standard *largest* slot size allows sharing between VOs without losing slot
  - SCORE still needed
    - fills ‘awkward’ #cores, not divisible by 16
    - draining 16 cores vs 8 : better than 2015, worse than now
- Reduced BS types to HTCondor & Slurm
  - experience in draining and keeping slots. Tunable loss depending on rate to acquire slots.
  - have walltime limit set, some short jobs for backfill

HT/SMT aside

- Cores left idle by memory scheduling or draining slots not really that bad
- Using 80 from 96 or 184/224 cores
  - gets the full node HS23
- That accounting is wrong is not a reason to obsess about idle cores
  - fix accounting, or live with it
- Some sites have HT-off or partial
  - rely on backfill. Revisit decision?
  - option to stay at 8 (higher HS23/core)





HS23 plots courtesy of Thomas Hartmann(DESY)

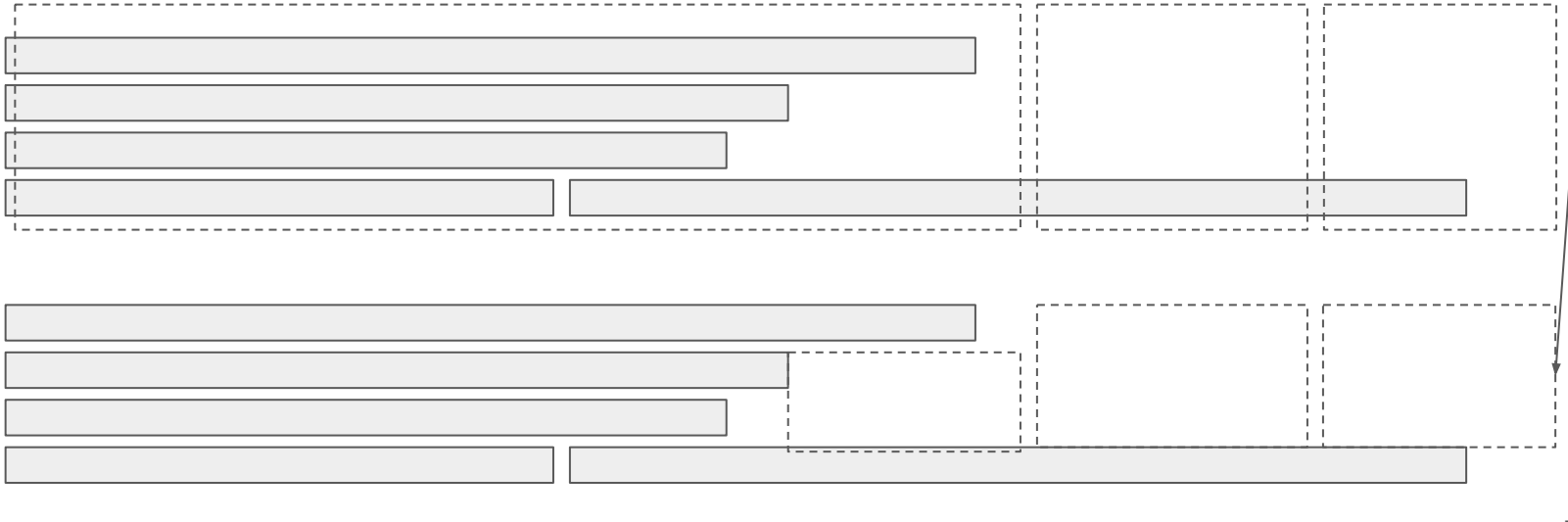
Whole node scheduling

- Required on most HPC resources
  - currently only running G4 simulation as not all MP/MT workloads scale perfectly
  - depending on future resource mix may need solution to pack node with smaller jobs to run all workloads
- Where serial batch system flexibility exposed via CE
  - no clear benefit or need for whole nodes
- Potential use-cases needing to freely schedule node resources
  - ML using all cores and shared memory
  - multiple processes offloading to GPU
- pledged resources continue with S/MCORE jobs
  - testing whole node on limited number of shared sites that support it for other VOs
    - to help with opportunistic use between VOs
    - we can run G4 Sim, or deploy/test overlayBS, e.g. Cobald-Tardis
      - need user-level cgroups v2 to avoid chaos of job interference

Ideal backfill without ideal job mix

```
requirements = Cpus >= 3 &&
(PartitionableSlot || Cpus <= 8)
request_cpus = Cpus > 8 ? 8 : Cpus
request_memory = Cpus * 2000
rank = Cpus
```

- Endless mix including short SCORE could keep full during draining
- Single job that can efficiently use the whole node
  - use all cores with nice'd (or zero cgroup share) background job
  - or just the free ones: I'd like at least 3 cpus, up to max of 8 if available



Conclusions

- Allow jobs to request memory over physical amount per core
  - pledged hardware request continues to be 2GB/core
  - trust and verify reasonable mix maintained, so no cores idle and no accounting problem
- Move to 16 core as standard on shared sites where a VO requests it
  - Sites can choose to stay with 8. Only important that VOs use the same per site.
- Whole node scheduling
  - pledged resources continue with S/MCORE jobs
    - no problem with CMS/ALICE pledges being wholenode, but ideally have MCORE too
  - keep opportunistic usage possible when VO idle, in both directions
    - needs expert Batch System config + backfill jobs with walltime limit
  - ATLAS wholenode: only G4 sim. Devel for potential future resource mix (HPC, Cloud)
  - foresee a limited number of volunteer sites

Back up

Accounting

- What if we have high priority tasks and willing to leave cores idle
  - easy: don`t maintain job mix
  - unhappy site admins would need accounting solace.
- Current accounting is core HS23 * walltime seconds
  - site wants full HS23 accounted when cores full OR RSS full, i.e. a missing dimension
  - reserve 2 cores for 4GB serial job? Works but don`t, because we only use 1 core
    - someone else(maybe same VO) can use that core
- Minimal change is not to add dimension but define an effective HS23s
- sum(job.ramcount)/sum(job.corecount) /site.meanrss, over running jobs
  - <= 1: account all jobs with full HS23/core as usual
  - >1 means cores *could* be idle.
    - effective HS23 scaled by requested RSS per core / site mean RSS
  - Jobs effectively using more than 1 core, but some using less than 1 - **not** an integer