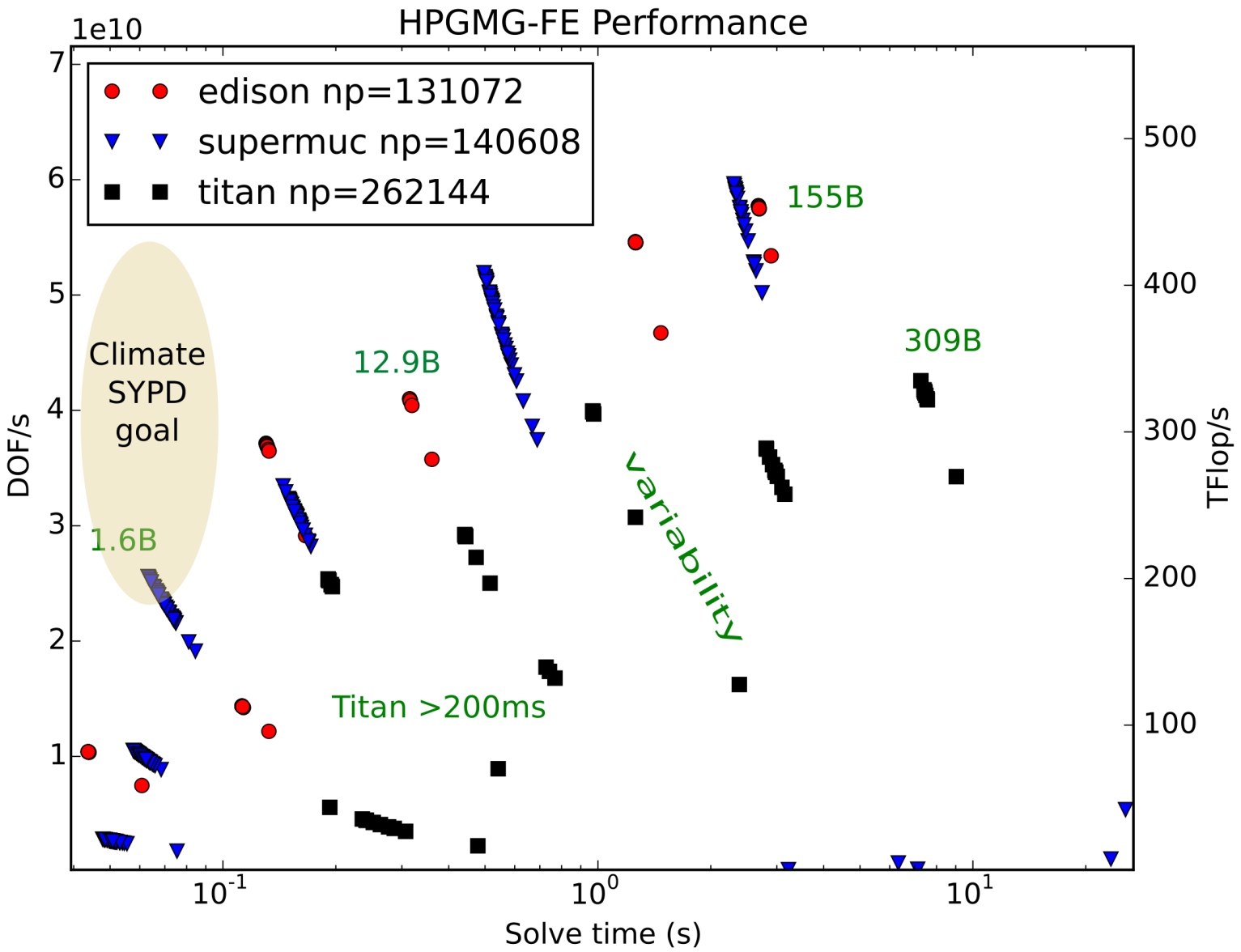
To Thread or Not To Thread

Jed Brown
Collaborators: Barry Smith, Karl Rupp, Matthew Knepley, Mark Adams, Lois Curfman McInnes

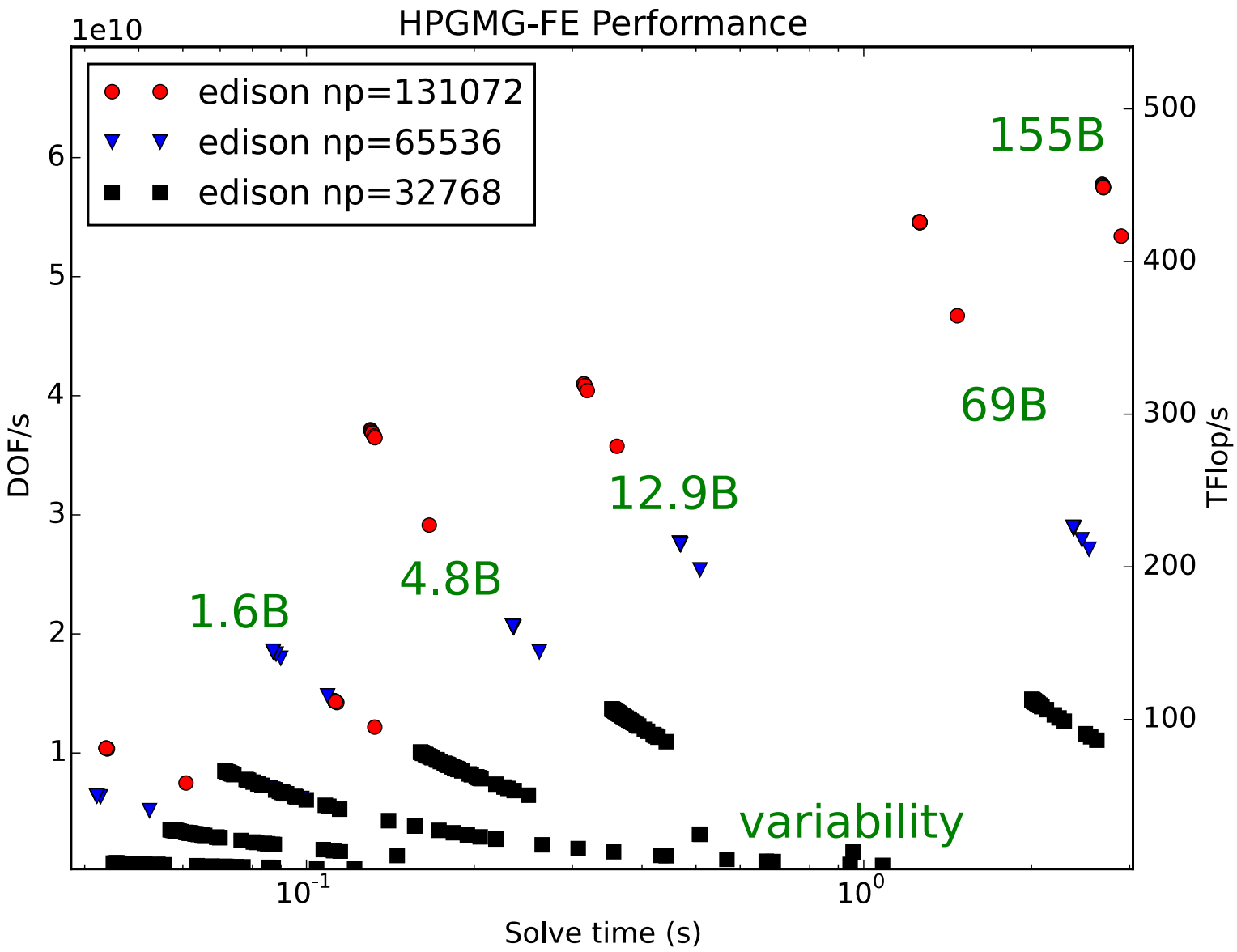
CU Boulder and Argonne National Laboratory

Multi-core 5 Workshop, 2015-09-16

Scaling regime: HPGMG-FE on Edison, SuperMUC, Titan



Scaling regime: HPGMG-FE on Edison at various scales



CAM-SE dynamics numbers

- 25 km resolution, 18 seconds/RK stage
- Current performance at strong scaling limit

Edison	3 SYPD
Titan	2 SYPD
Mira	0.9 SYPD

- Performance requirement: 5 SYPD (about 2000x faster than real time)
  - 10 ms budget per dynamics stage
  - Increasing spatial resolution decreases this budget
- Null hypothesis: Edison will run ACME faster than any DOE machine through 2020
  - Difficult to get large allocations

Party line

- Processes are heavy abstractions compared to threads
- Halo exchange is expensive – sharing is better
- OpenMP is lighter weight than MPI
- Processes have substantial memory overhead

Question

What is the difference between a thread and a process?

Question

What is the difference between a thread and a process?

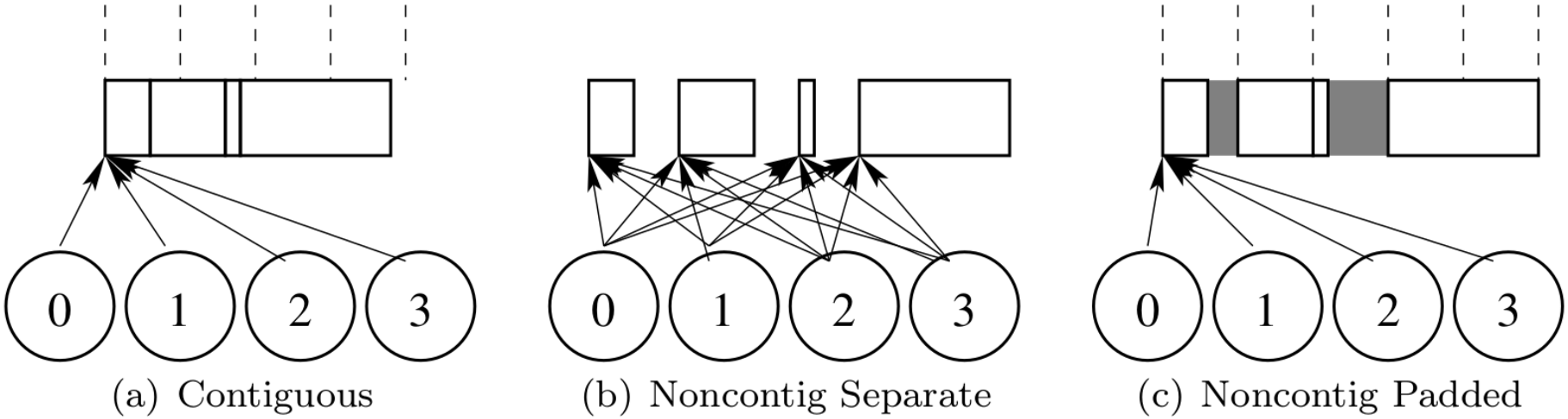
- Both are created using `clone(2)`
- Equivalent entries in kernel data structure
- Threads use `CLONE_VM`, processes have copy-on-write
- Rule of thumb
  - Threads cost 10μs to create
  - Processes cost 100μs to create
  - No difference in context switching
  - Only paid once – everyone uses thread pools anyway

Portable shared memory between MPI processes

- MPI-3 portable shared memory windows
- `MPI_Comm_split_type(comm, MPI_COMM_TYPE_SHARED, 0, MPI_INFO_NULL, &newcomm);`
- `int MPI_Win_allocate_shared(MPI_Aint size, int disp_unit, MPI_Info info, MPI_Comm comm, void *baseptr, MPI_Win *win);`

[Hoefler et al, MPI+MPI, 2013]

Halos or contiguous memory?



- Common assumption: halo copying is expensive
- Alternative is shared memory
- Cache utilization for 16^3 local domain with halos
  - Entire local region is contiguous; no partially filled cache lines
  - $18^3 * \texttt{sizeof(double)} = 46656B$
- 16^3 local domain embedded in contiguous memory
  - Avoid false sharing: align owned portion to cache-line boundaries
  - $32 \times 18 \times 18 * \texttt{sizeof(double)} = 82944B$
  - False sharing a serious problem if local sizes not divisible by line size

Messaging from threaded code

- Off-node messages need to be packed and unpacked
- Many MPI+threads apps pack in serial – bottleneck
- Extra software synchronization required to pack in parallel
  - Formally $O(\log T)$ critical path, T threads/NIC context
  - Typical OpenMP uses barrier – oversynchronizes
- `MPI_THREAD_MULTIPLE` – atomics and $O(T)$ critical path
- Choose serial or parallel packing based on T and message sizes?
- ≥ 1 hardware NIC context/core now, maybe not in future
- What is lowest overhead approach to message coalescing?

But processes can't work for hyperthreads (?)

- Can processes hyperthreaded onto the same core share L1 cache?
- Modern caches are physically tagged
  - Identical cache sharing to threads
- TLB is not shared between processes
  - Is your application TLB-limited?

But processes can't work for hyperthreads (?)

- Can processes hyperthreaded onto the same core share L1 cache?
- Modern caches are physically tagged
  - Identical cache sharing to threads
- TLB is not shared between processes
  - Is your application TLB-limited?

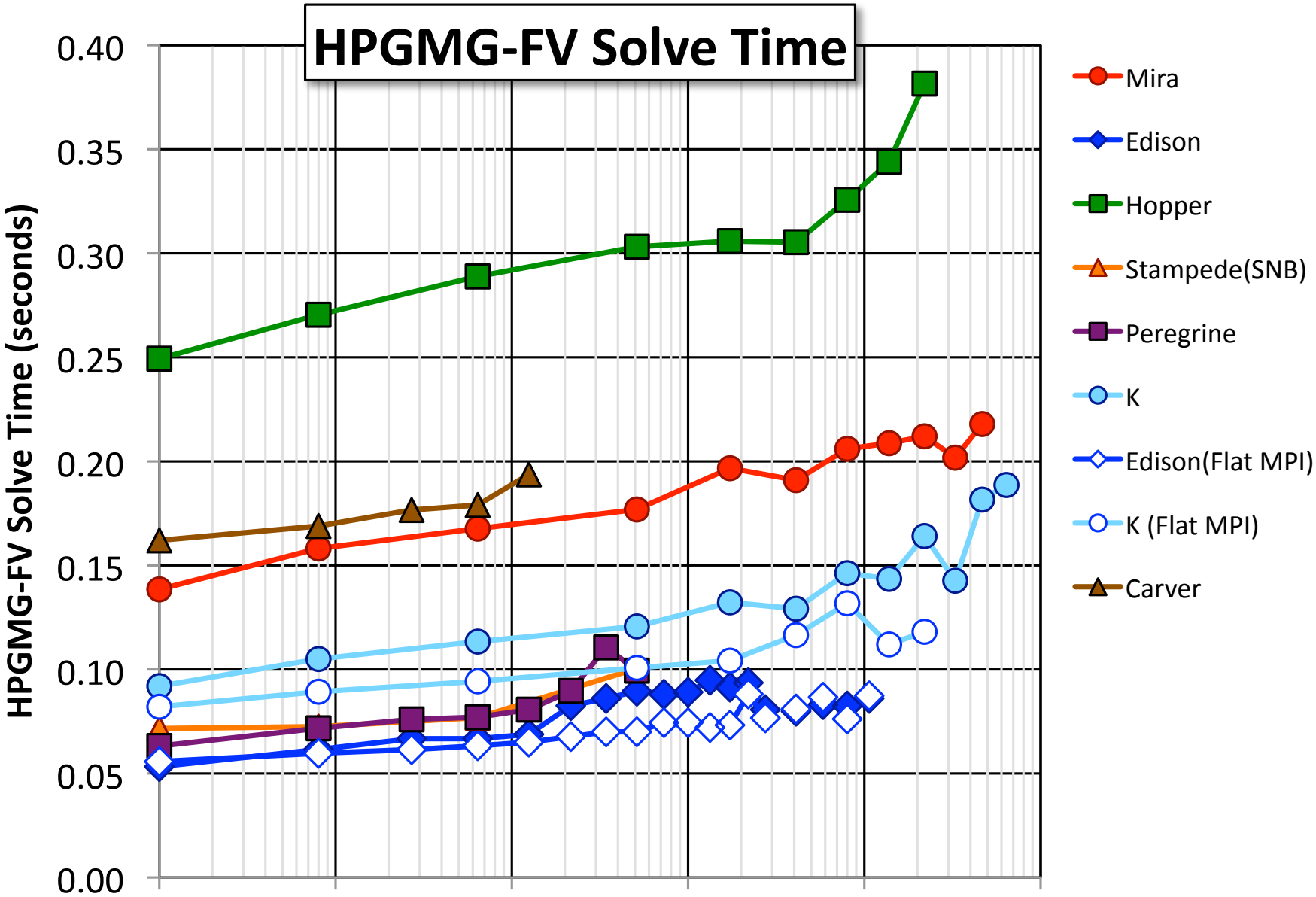
But processes can't work for hyperthreads (?)

- Can processes hyperthreaded onto the same core share L1 cache?
- Modern caches are physically tagged
  - Identical cache sharing to threads
- TLB is not shared between processes
  - Is your application TLB-limited?

Sharing large read-only data/code

- Templated/generated code
- Lookup tables
- Allocate dynamically in a shared window
- Compile into shared library: transparently shared

HPGMG-FV: flat MPI vs MPI+OpenMP (Aug 2014)



Outlook

- Application scaling mode must be scientifically relevant
- Threads and processes are more alike than usually acknowledged
- Processes versus threads is about shared versus private by default
  - No problem to share when desirable
  - Debuggability consequences
- Algorithmic barriers exist
  - Throughput architectures are not just “hard to program”
- Vectorization versus memory locality
- What is the cost of performance variability?
  - Measure best performance, average, median, 10th percentile?