Advanced OpenMP

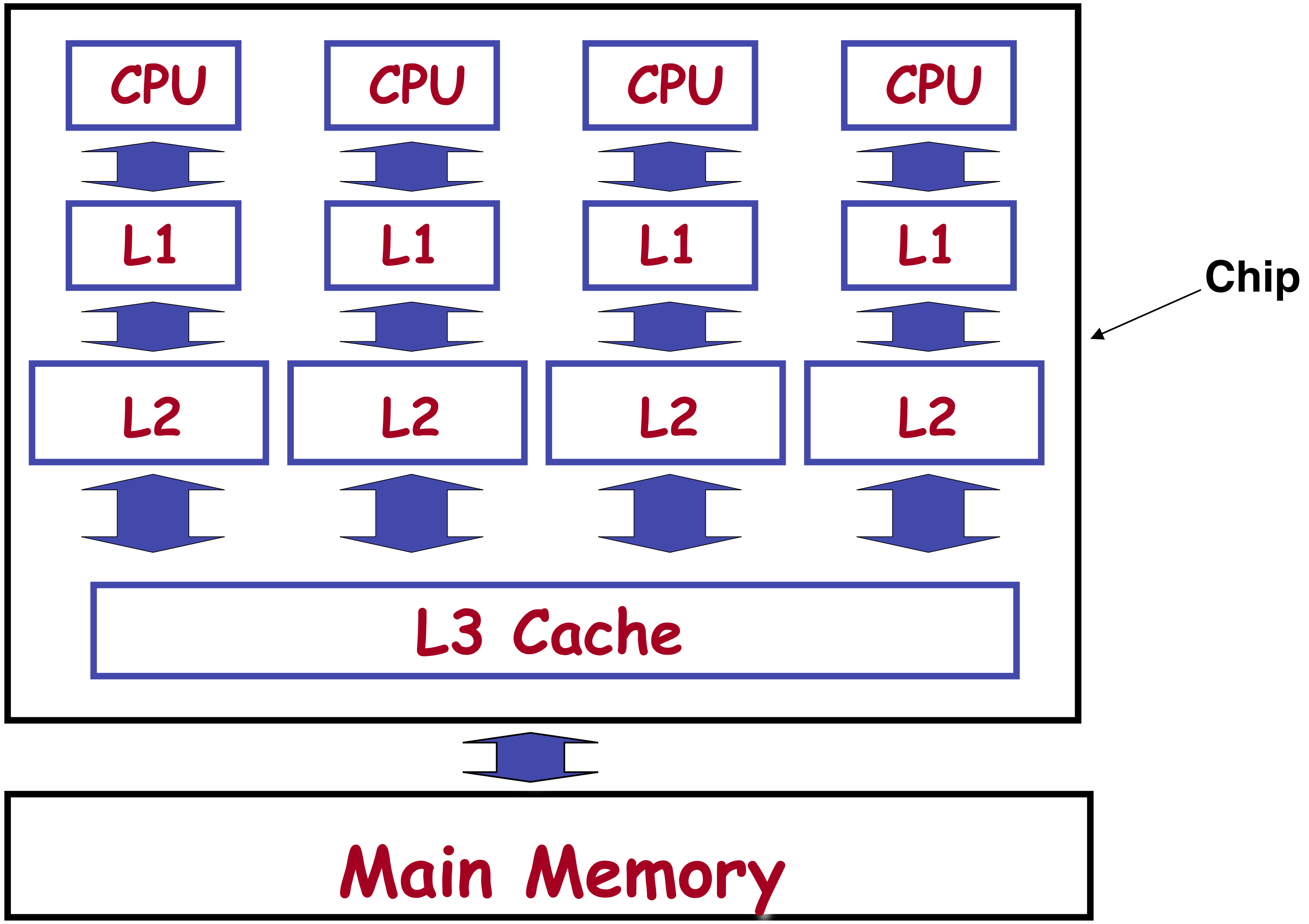
Lecture 2:

Multicore and multithreaded processors

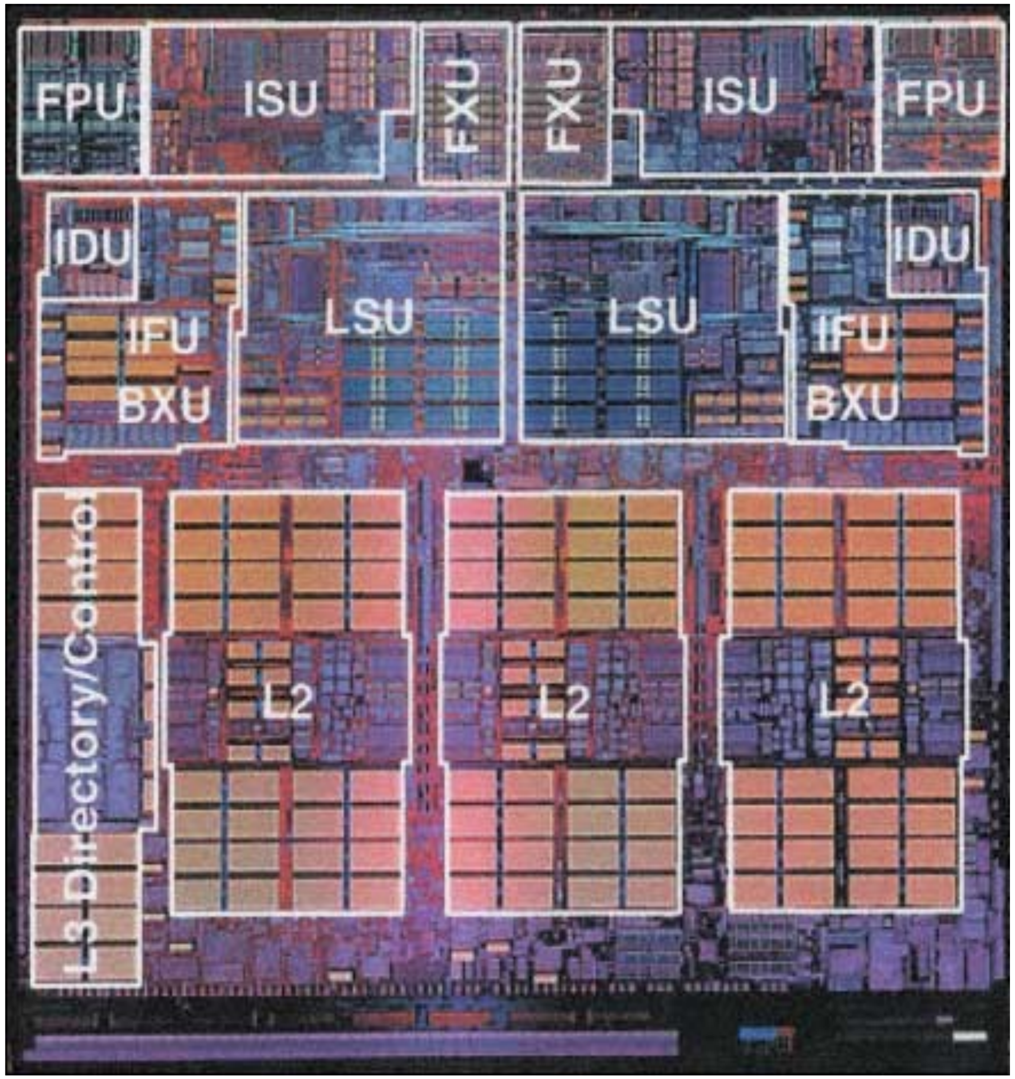
Multicore chips

- Now possible (and economically desirable) to place multiple processors on a chip.
- From a programming perspective, this is largely irrelevant
  - simply a convenient way to build a small SMP
  - on-chip buses can have very high bandwidth
- Main difference is that processors may share caches
- Typically, each core has its own Level 1 and Level 2 caches, but the Level 3 cache is shared between cores

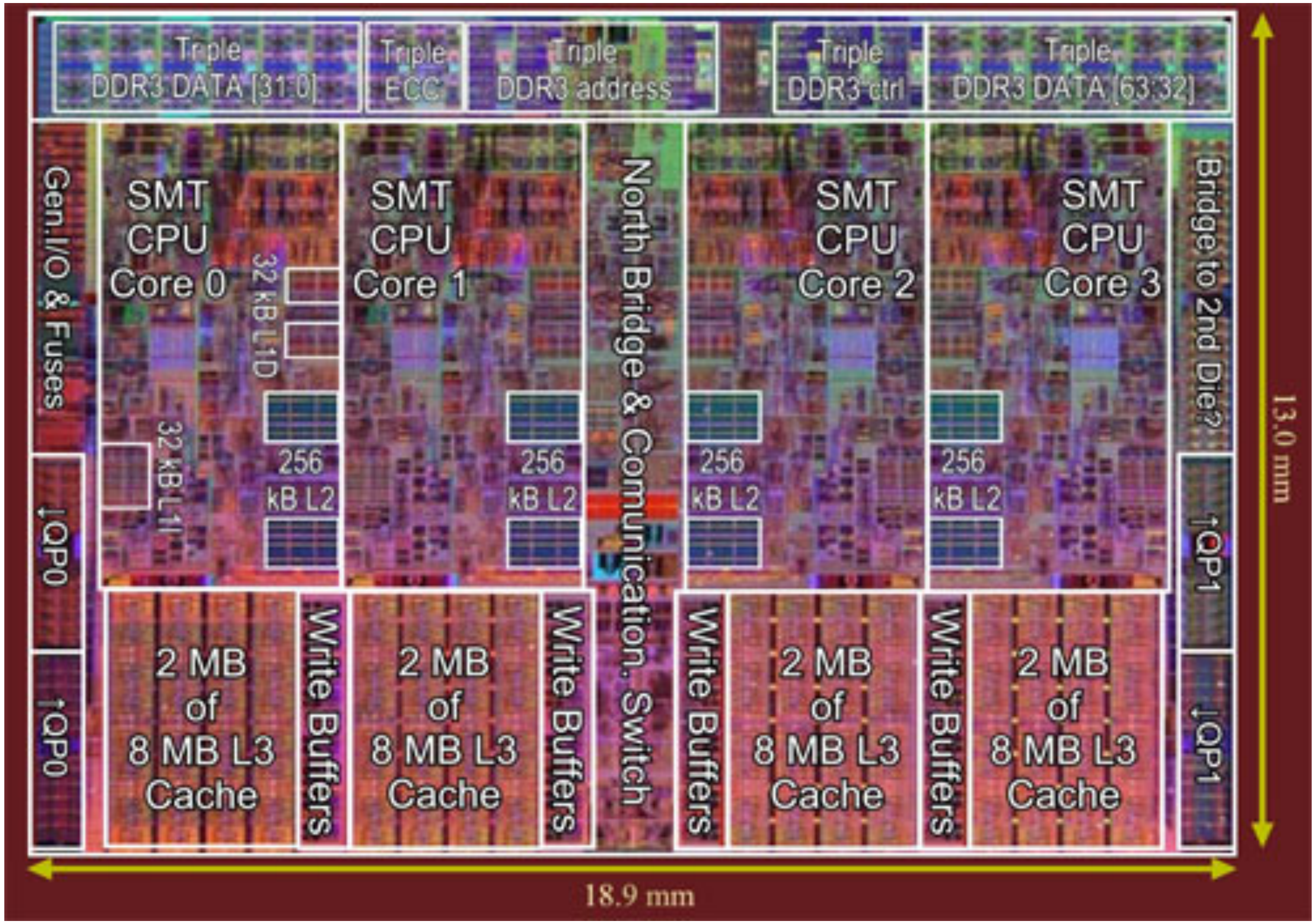
Typical cache hierarchy



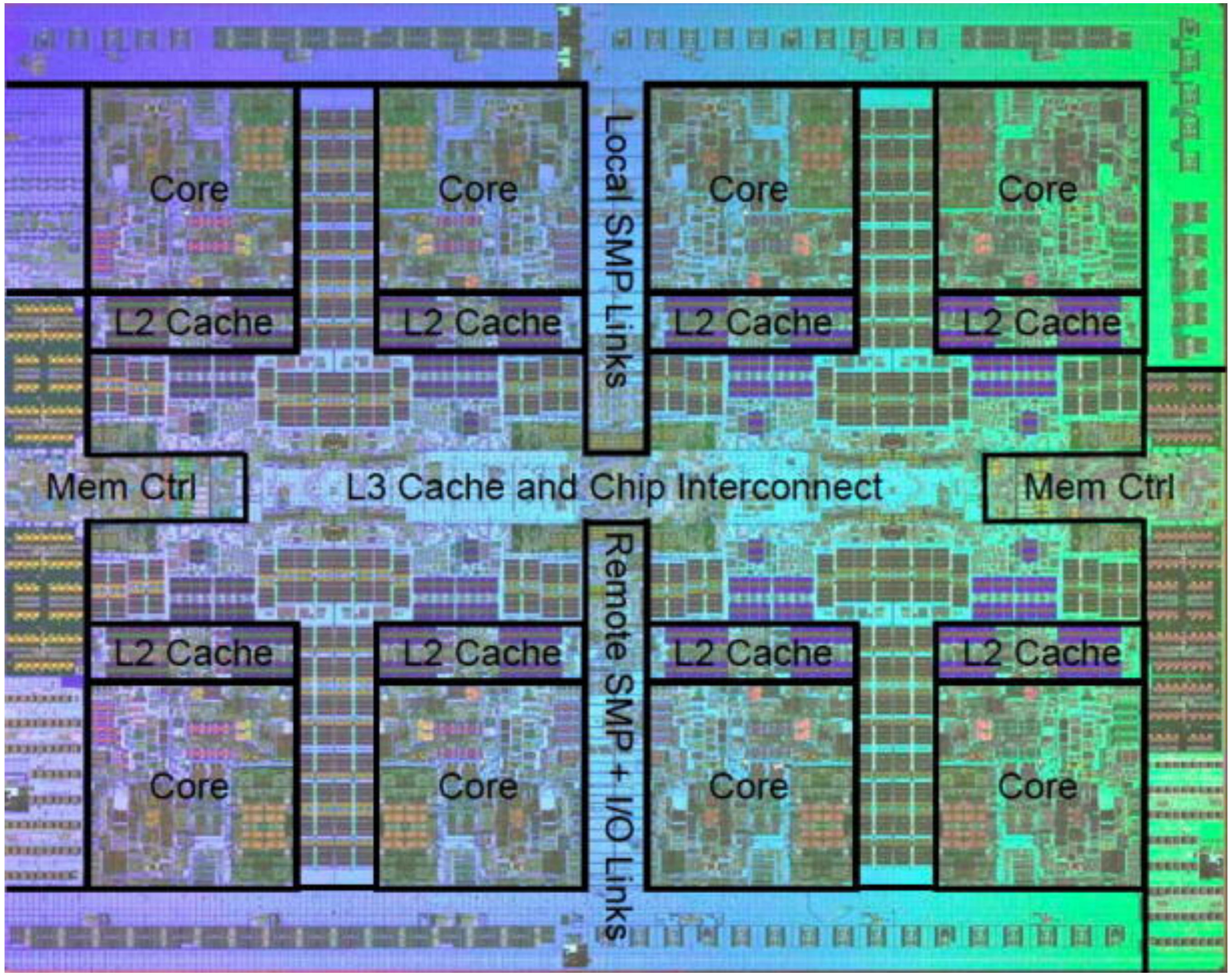
Power4 two-core chip



Intel Nehalem quad-core chip



Power7 8-core chip



- This means that multiple cores on the same chip can communicate with low latency and high bandwidth
  - via reads and writes which are cached in the shared cache
- However, cores contend for space in the shared cache
  - one thread may suffer capacity and/or conflict misses caused by threads/processes on another core
  - harder to have precise control over what data is in the cache
  - if only single core is running, then it may have access to the whole shared cache
- Cores also have to share off-chip bandwidth
  - for access to main memory

Latency hiding with multiple threads

- A processor may frequently stall while a memory access is in progress
- Better use of the processor may be made by running another thread in the “gap”
  - latency hiding
- Cannot be done with standard multitasking
  - cost for a context switch by the OS is ~1000s of cycles
  - longer than a main memory access

Conventional multithreading

- With hardware support, a thread switch can be done in a single clock cycle
  - may need to have multiple register files, one for each thread
- Can simply round-robin threads on consecutive cycles, or switch when a thread stalls on a load.
- Extreme example is the Cray XMT
  - 128 threads per processor
  - no data caches
  - typical applications require 10-20 threads per processor to hide memory latencies
- Also used in Intel Xeon Phi
  - ~60 cores, 4 threads per core

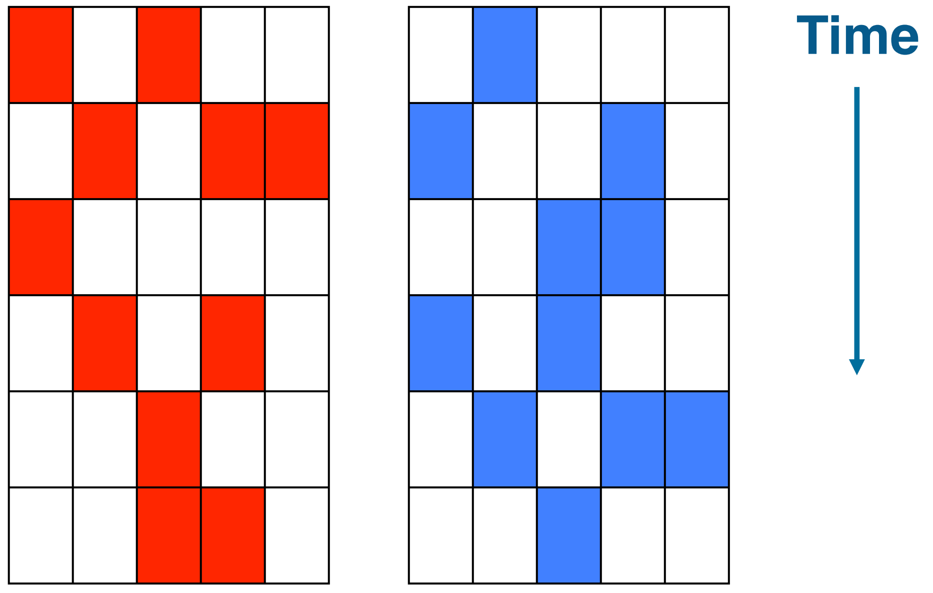
Empty instruction slots

- Most modern processors are superscalar
  - can issue several instructions in every clock cycle
  - selection and scheduling of instructions is done on-the-fly, in hardware
- A typical processor can issue 4 or 5 instructions per clock, going to different functional units
  - obviously, there must be no dependencies between instructions issue on the same cycle
- However, typical applications don’t have this much instruction level parallelism (ILP)
  - 1.5 or 2 is normal
  - more than half the available instruction slots are empty

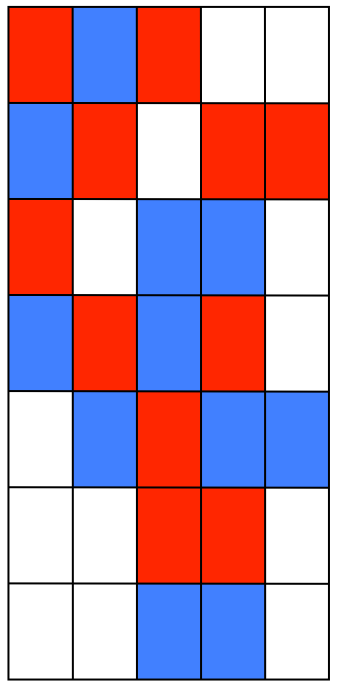
SMT

- Simultaneous multithreading (SMT) (a.k.a. Hyperthreading) tries to fill these spare slots by mixing instructions from more than one thread in the same clock cycle.
- Requires some replication of hardware
  - instruction pointer, instruction TLB, register rename logic, etc.
  - Intel Xeon only requires about 5% extra chip area to support SMT
- ...but everything else is shared between threads
  - functional units, register file, memory system (including caches)
  - sharing of caches means there is no coherency problem
- For most architectures, two or four threads is all that makes sense

SMT example



Two threads on two CPUs



Two threads on one SMT CPU

More on SMT

- How successful is SMT?
  - depends on the application, and how the 2 threads contend for the shared resources.
- In practice, gains seem to be limited to around 1.2 to 1.3 times speedup over a single thread.
  - benefits will be limited if both threads are using the same functional units (e.g. FPUs) intensively.
- For memory intensive code, SMT can cause slow down
  - caches are not thread-aware
  - when two threads share the same caches, each will cause evictions of data belonging to the other thread.

Multicore vs. SMT

- Can view multicore and SMT as two extremes of a replication continuum
  - multicore replicates the entire CPU
  - SMT replicates as little as possible
- May in the future see something in-between
  - e.g. multiple cores which share some functional units
  - AMD Bulldozer core share a floating point unit
- Will seriously complicate the notion of how many processors there are in a system!
  - already a problem with SMT cores counted as two "virtual processors"