


Software Issues for Multicore Systems



Luddy Harrison

PPoPP 2006

What is new this time around?

- ↓ The applications that need it are not scientific, by and large
- ↓ The programmers who will do this do not hold advanced degrees, necessarily
- ↓ The software systems that will incorporate it are commercial products
- ↑ The cost of failure (inefficiency) is much lower

Only one of these (the first) is a *technical* consideration. The others are *economic* considerations.

What is required?

- We must separate once for all the development of *components* from their deployment as a *parallel system*
- In the development of hardware systems, we write HDL *modules* (read: components) that are *automatically deployed in parallel*
 - A parallel system is established by rules of composition
 - The correct functioning of components depends very little on their composition into a larger system
- Without such a principle in a multicore software development environment, we will never succeed
 - Parallel software systems *must be much simpler, architecturally, than sequential ones* if they have a chance of holding together
 - From a correctness point of view
 - From a performance point of view

FPGA development, by analogy

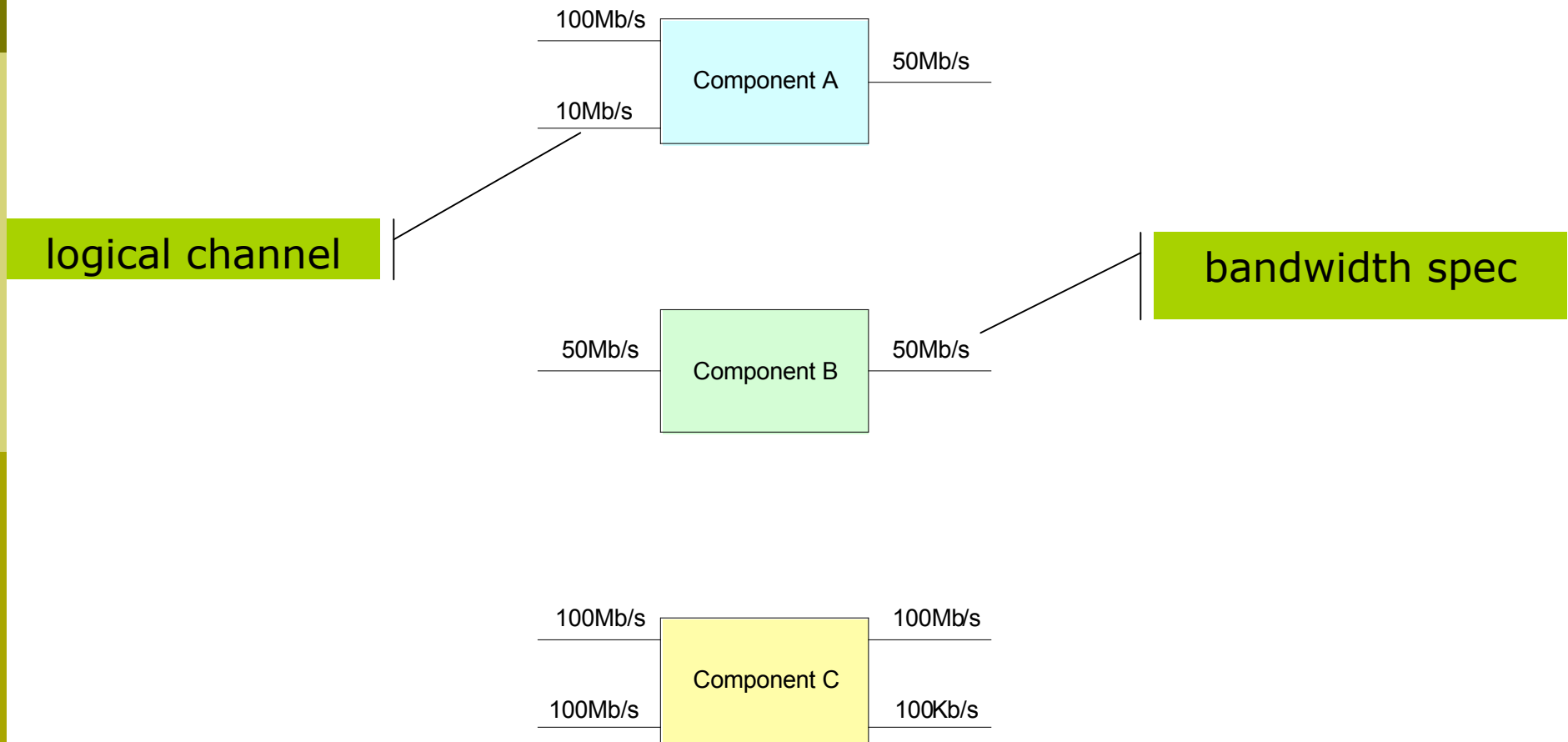
- The development of components for FPGAs is manual or semi-automatic
 - I draw schematics
 - I write Verilog
 - I write System C
 - I use a generator like Matlab or Spiral

- Their deployment onto the FPGA fabric is essentially automatic
 - The geometry of the FPGA is irrelevant or else relevant only occasionally (e.g., the placement of I/O functionality)
 - My components can be used on any vendor's FPGA
 - The components run in parallel. I don't have to think hard about this.

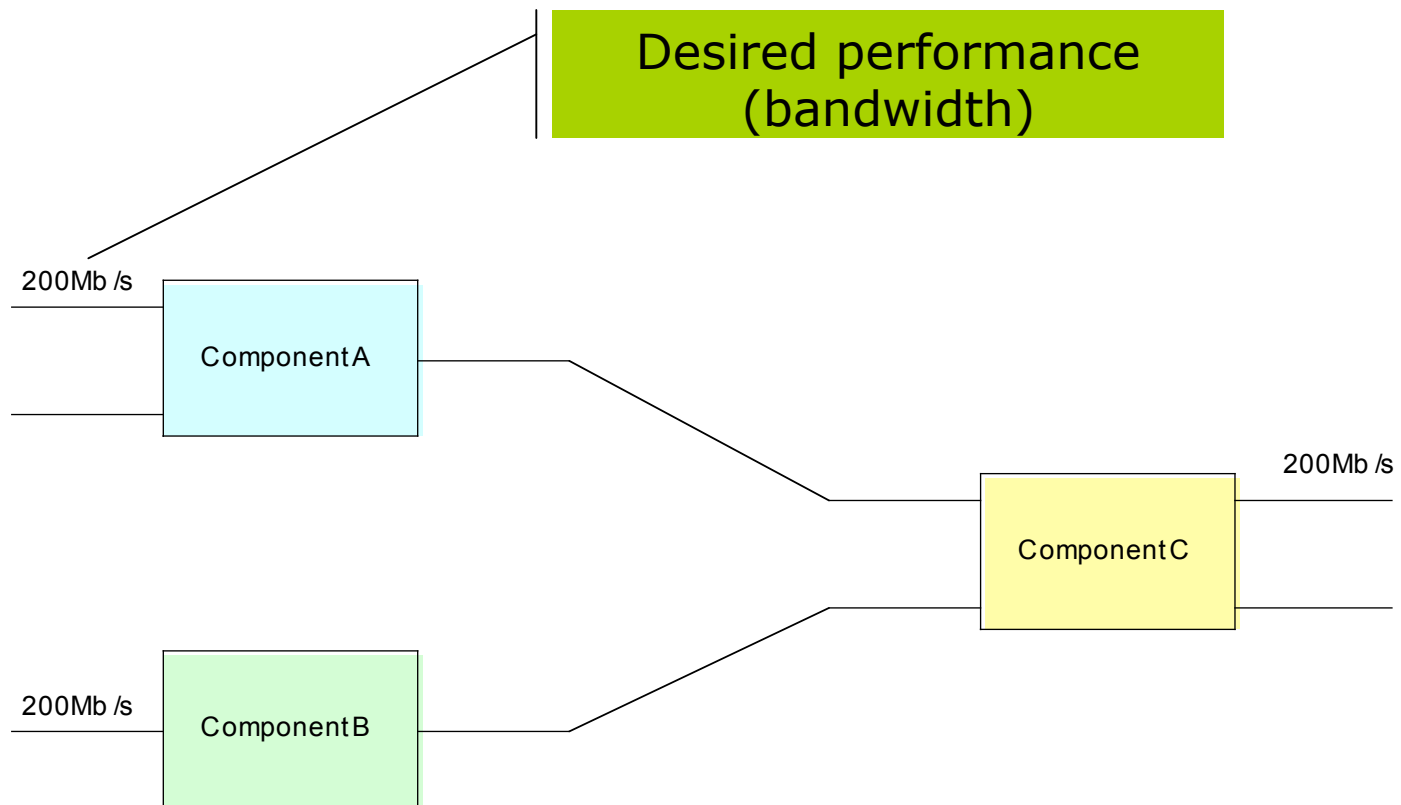
If we developed this way...

- We would write *components* that had a freestanding semantics
- A *place-and-route* tool would lay the components onto a multicore system and manage the interconnection between the cores
- Simple *rules of composition* would determine the correctness of the system based on the correctness of the components
 - And ideally, the *performance* as well as the *correctness*
- We don't have *any of these things* for parallel software systems today.
 - No established component model / interface contracts
 - No established place-and-route technology or standard interface between components and such technology
 - No simple rules of composition used to establish large-scale parallel systems

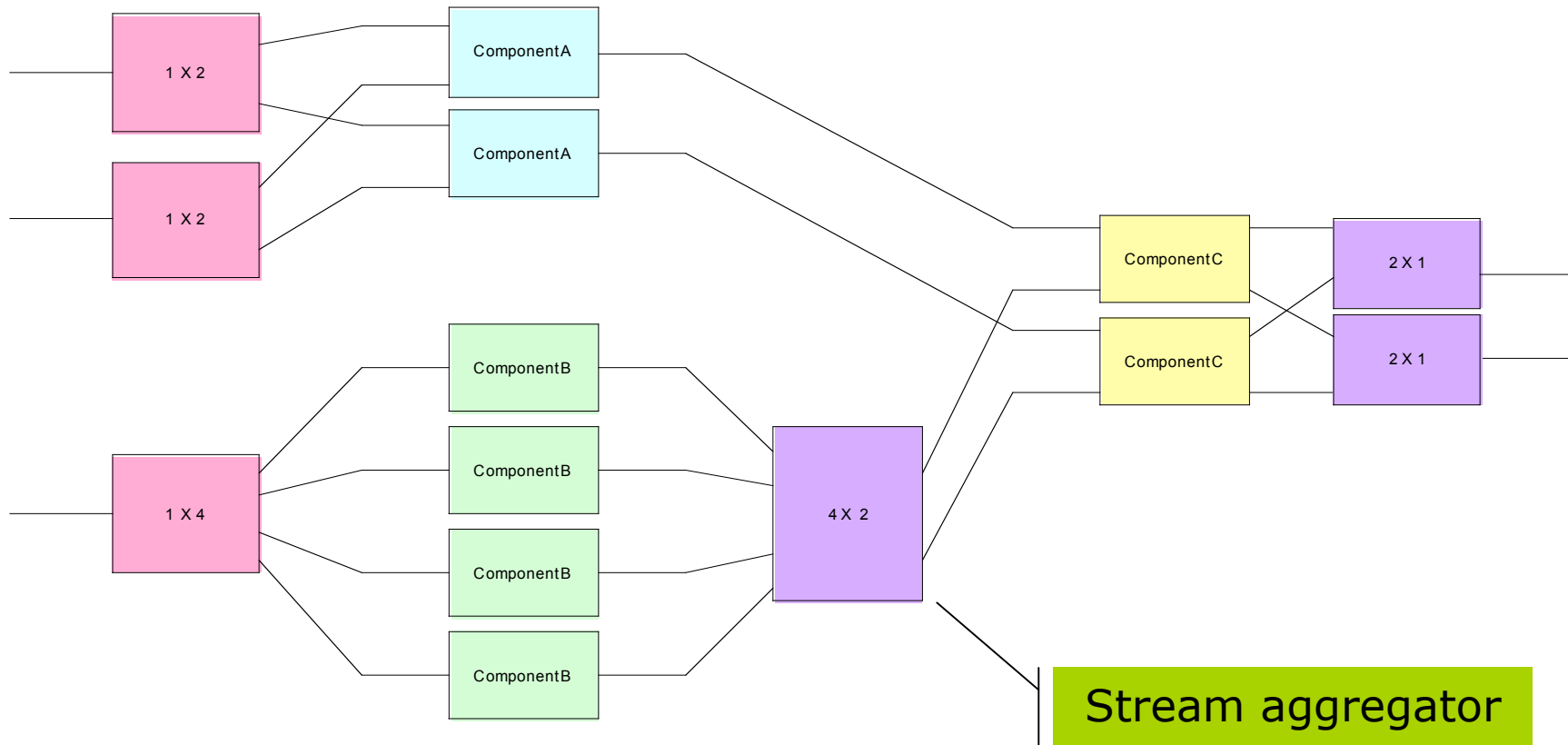
A Component Library



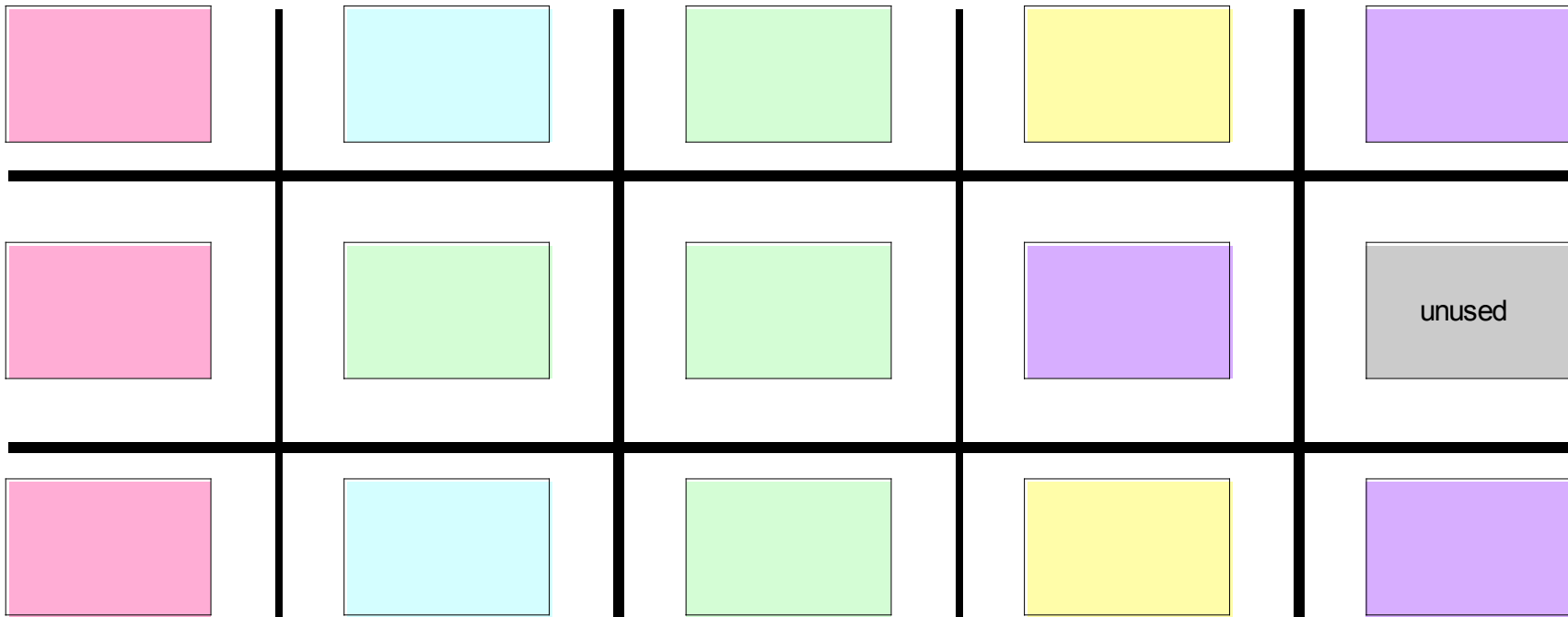
An Application



A Realization of the Performance



A Mapping



Channel bandwidth
Managed by mapper

Prediction Concerning Caches

- We will struggle before finally accepting that *the cache abstraction does not scale*
 - Efficient point-to-point communication is required
 - Caches cannot successfully fake this
 - But caches have considerable momentum
 - They seem to allow global shared memory (SW)
 - They fit well with multithreaded models (SW)
 - They don't require us to rigorously isolate communication (SW)
 - They are already built into processor cores (HW)
 - They won't go away quietly or quickly
 - Therefore most success will be achieved on *nonstandard* multicore platforms like graphics processors, network processors, signal processors, where there is less investment in caches.