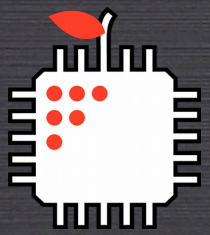
# Heterogeneous integration

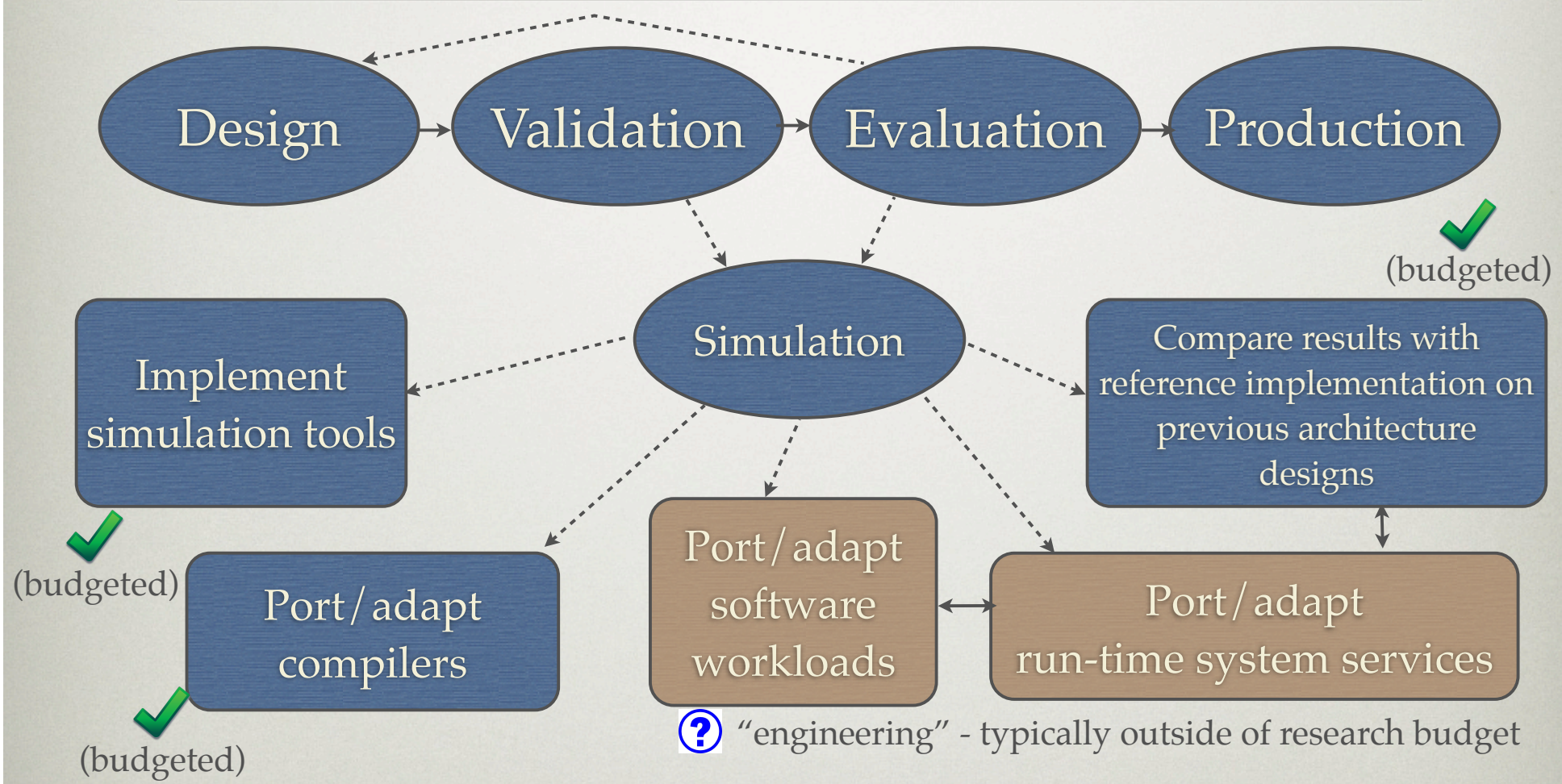
## to simplify many-core architecture simulations

Raphael 'kena' Poss
University of Amsterdam



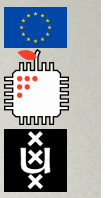
RAPIDO - HiPEAC 2012, Paris

# Hidden costs in Processor innovation




**Assumption that existing codes can be "ported" with minimal work** → **What to do when a new design makes porting fundamentally difficult?**

# Funding priorities in architecture research



- **Innovation happens "on the chip"**
  - Requires focussed effort to design and simulate interactions between many cores and caches
  - *Funding difficult to obtain for "around the chip" efforts*
- **"Features" required by legacy OSs distract from this focus**, especially if mandated on every core; e.g.:
  - System description tables in ROM
  - Backward-compatible PICs and IPIs
  - Shared I/O and memory address space
  - Standard traps (FPU exceptions, syscalls, debugging)

# Reusing software stacks with newly designed Many-P-SoCs

- **Uniprocessor/SMP software stacks**
  - control flows back and forth between “system” and “application” within a single instruction stream (“syscalls”)
  - **require all system services on all cores**
- **Distributed/cluster stacks**
  - dependent on full network stacks between processor+RAM nodes
  - large latency and buffering overheads, **unsuitable for low-latency on chip communication** between cores
- **Experimental many-core OSs (eg Barrelfish, fos, ...)**
  - research projects on their own, may not be willing to divert effort away from their own grant focus
  - Still little compatibility with existing benchmark workloads

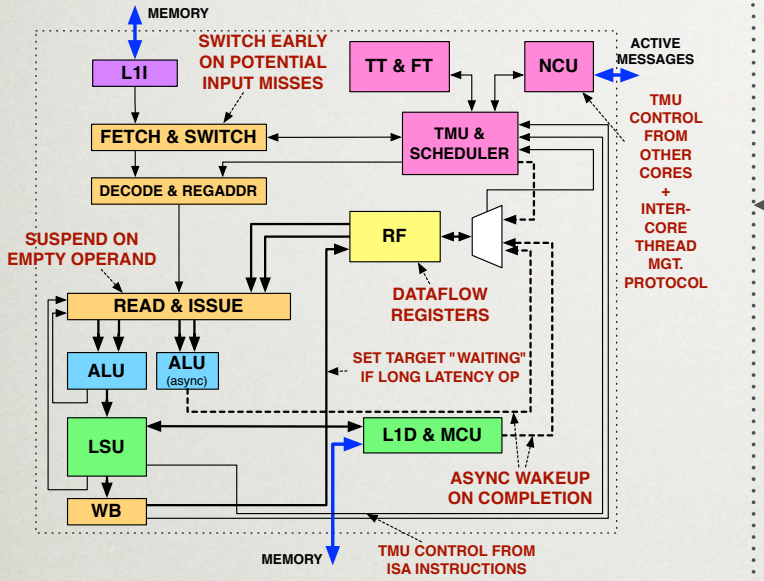
# Known strategies

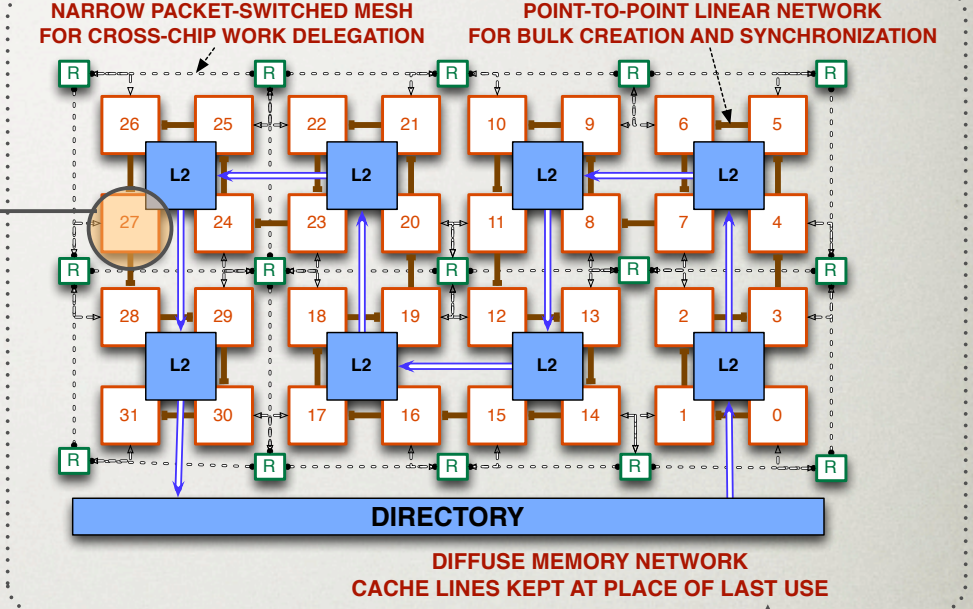
- Design **incrementally** to keep the Uni/SMP abstractions
  - i.e. new features but (mostly) *same semantics*
  - "more of the same" - might get stuck in local minima
- Advertise and design the new technology as "**accelerator**"
  - i.e. a form of *extension device*, eg. GPGPUs
  - keeps the innovation as a "side-kick" of legacy systems
- Extend the budget; **co-design the entire software stack**
  - *risky* and/or *too long to market* except for embedded
- Architecture research needs **radical new solutions, now**
  - we must *lower the cost to entry* to innovation

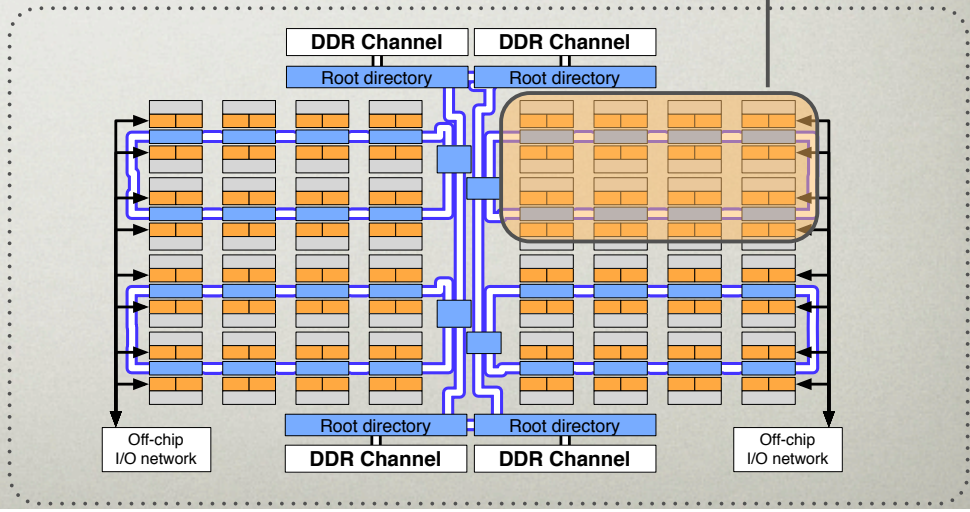
# Example fundamental innovations

- Processor design:
  - replace *interrupts* by *active messages*
  - replace *registers* by *dataflow synchronizers*
  - allow programs to *choose the number of registers* in the ISA *at run-time*
- Memory network design:
  - replace *cache coherency* by *explicit bulk consistency*
  - *share the address translation* between multiple cores
- These are merely examples, yet all have been explored in Apple-CORE and its Microgrid design.

# Our use case
# Microgrids of DRISC cores



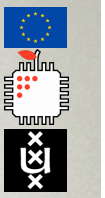




Many “new” properties, little “back” compat.
... can’t reuse existing software stacks as-is

**Should we abandon this direction just because the cost to port the entire software stack is huge?**

# Heterogeneous integration...

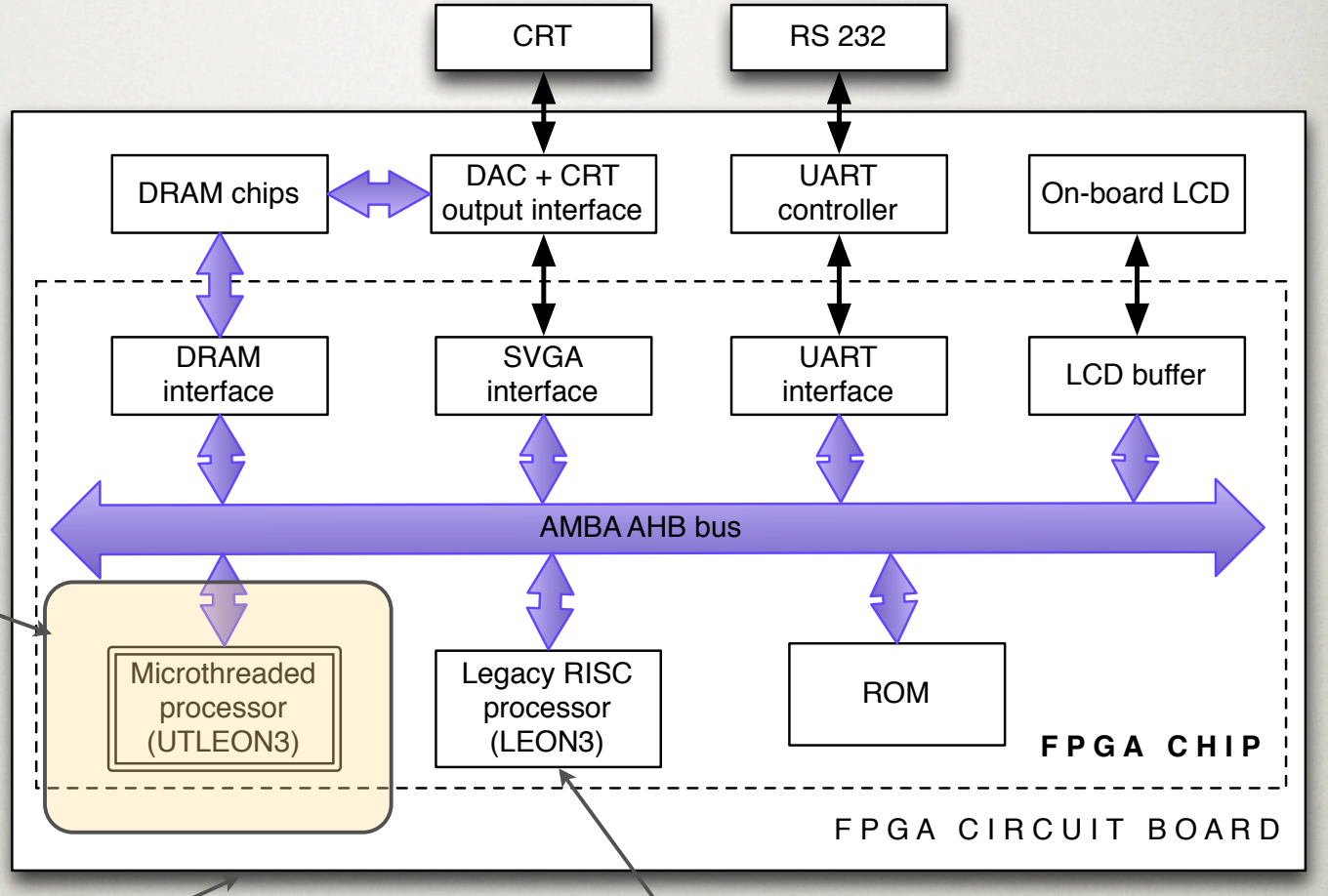


- Strategy to:
  - **support evaluation** of new systems
  - keep **focus on architecture research**
  - allow **freedom to "radically" innovate**
- Key idea: a many-core chip is a distributed system; use some nodes (cores) as **"service processors"** for legacy tasks, in particular console I/O, file access and/or job input
- Automatically redirect API calls through the simulated NoC or host/fabric interface in library wrappers
- Concept common in HPC, e.g. Cray XMT

# ...TO SIMPLIFY SIMULATION

- Different components can be simulated with different levels of accuracy
- In particular simulation inaccuracy in seldom used "service components" may be acceptable
- Integration in simulators becomes cheap because **service components can be functionally emulated**
- Can increase accuracy by either:
  - increasing accuracy of legacy component (short term)
  - incrementally migrating system services from legacy component to new components (long term)

# Example: UTLEON3 Microthreads on FPGA



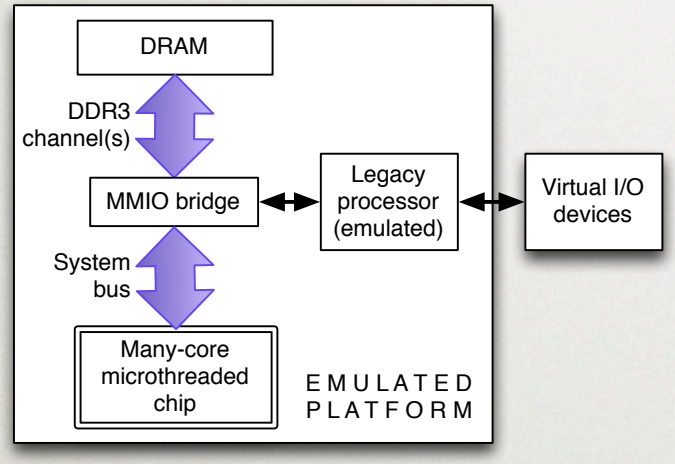

Research focus

Entire system required for validation/evaluation

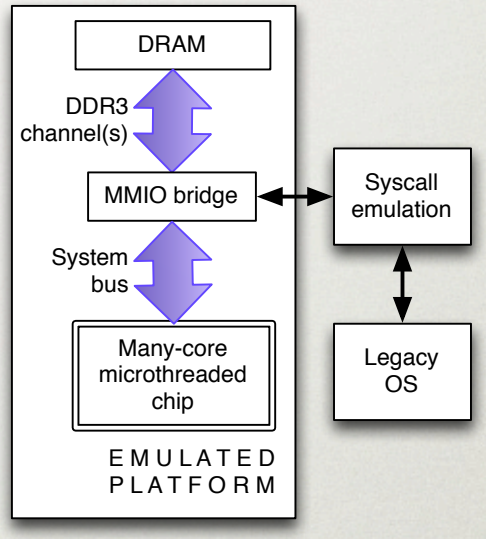
Operating system can run here

# Design points with software simulators

Where are service cores simulated?



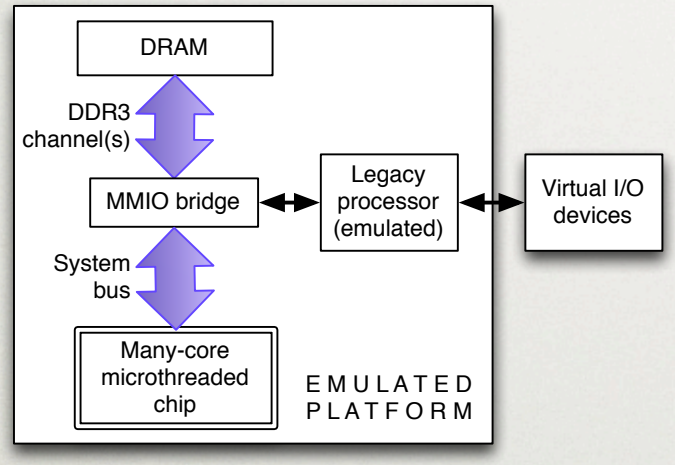

within the detailed simulation framework

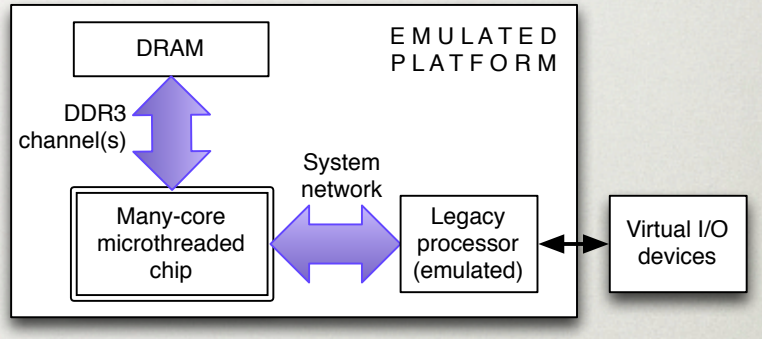



in a higher-level environment or natively on the simulation host

# Design points with software simulators

Where is the connection point?




Common NoC for memory and I/O

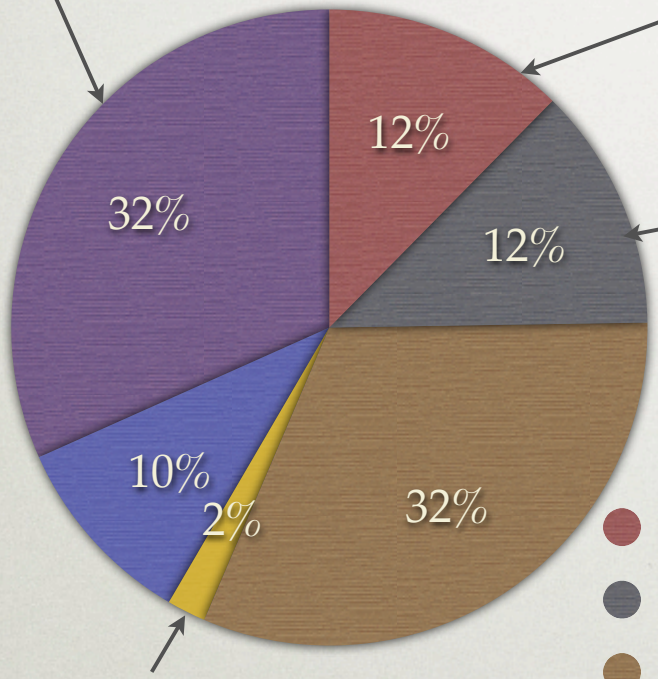
Separate NoC for memory and I/O

# Resulting distribution of engineering efforts

Focus on exploitation
of fine-grained, massive concurrency

2 simulators: FPGA + software

D-RISC core
L1 split-phase cache
COMA network

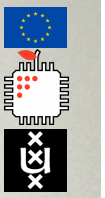



Can reuse existing
OS components on
legacy core

- Implementation of simulation components
- Low-level/RTL hardware design
- Low-level testing and debugging
- Porting system services
- Compiler back-end - machine code generation
- Higher-level evaluation (compilers & applications)

# Take away & future work

- Architecture research requires realistic full-system simulations, but full systems are expensive to reproduce;
  **heterogeneous integration reduces this cost**
- Heterogeneous integration **combines** naturally **with heterogeneous simulation** (e.g. FAST, MICRO'07)
- Various **integration strategies** allow to control the simulation accuracy of delegated services



# Thank you!