SCAF:
Scheduling and Allocation with Feedback

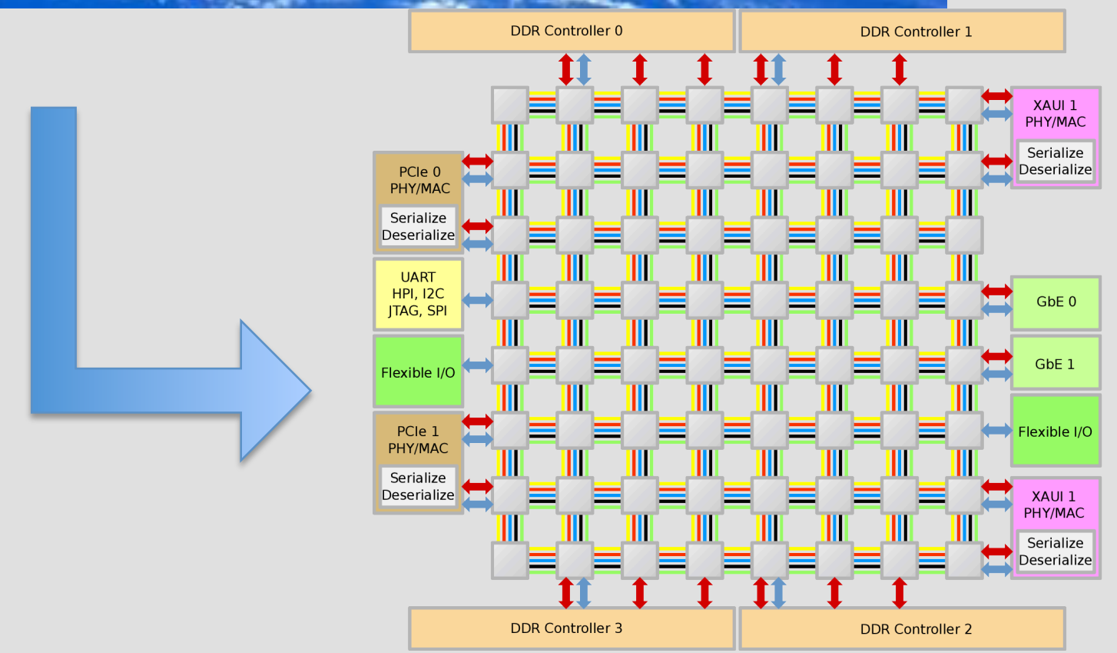
Tim Creech, Rajeev Barua

THIRTY YEARS OF SYSTEMS RESEARCH EXCELLENCE

INSTITUTE FOR SYSTEMS RESEARCH
A. JAMES CLARK SCHOOL OF ENGINEERING

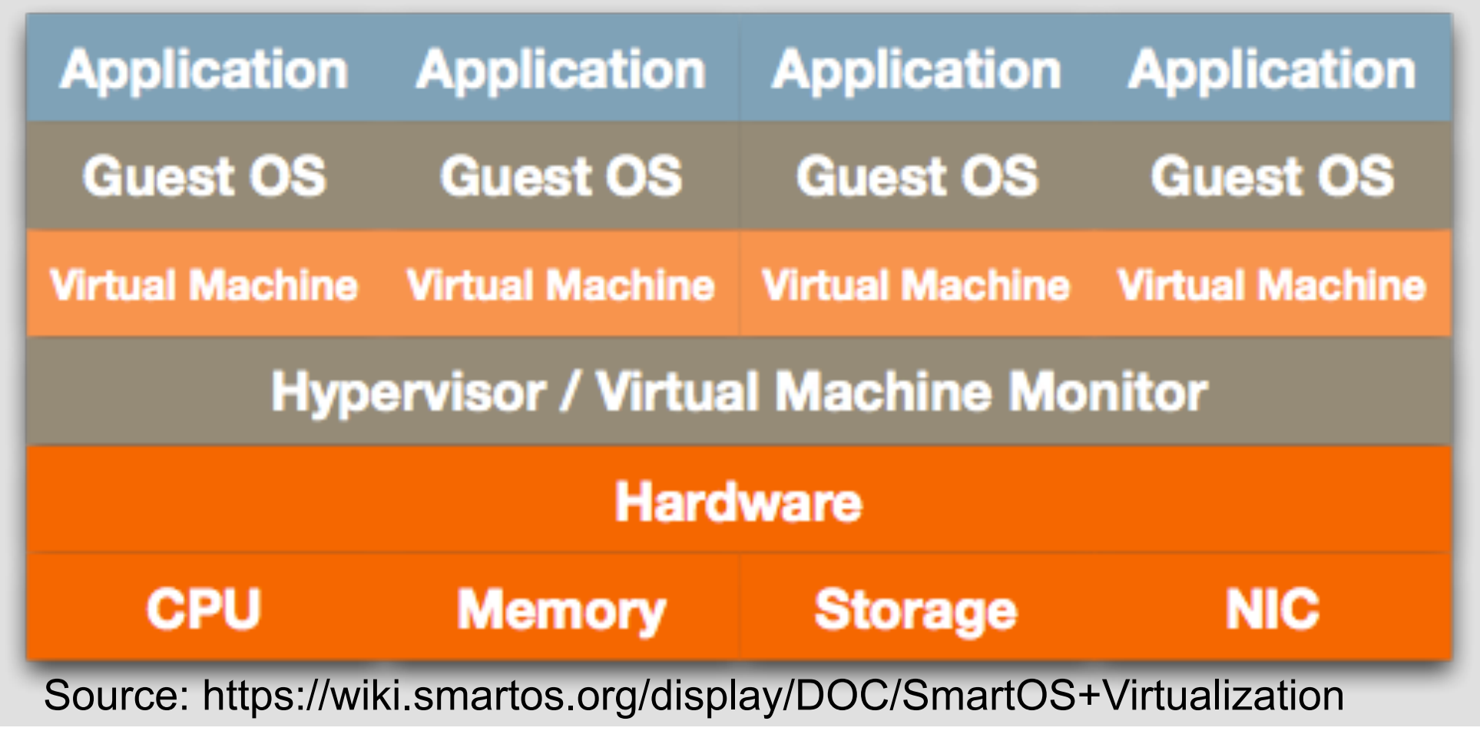
Inspired by work on AESOP, a parallelizing compiler for next-generation many-core satellites

- Need parallel programs for performance
- Need to run multiple programs
- ***How can we manage multiple parallel programs?***

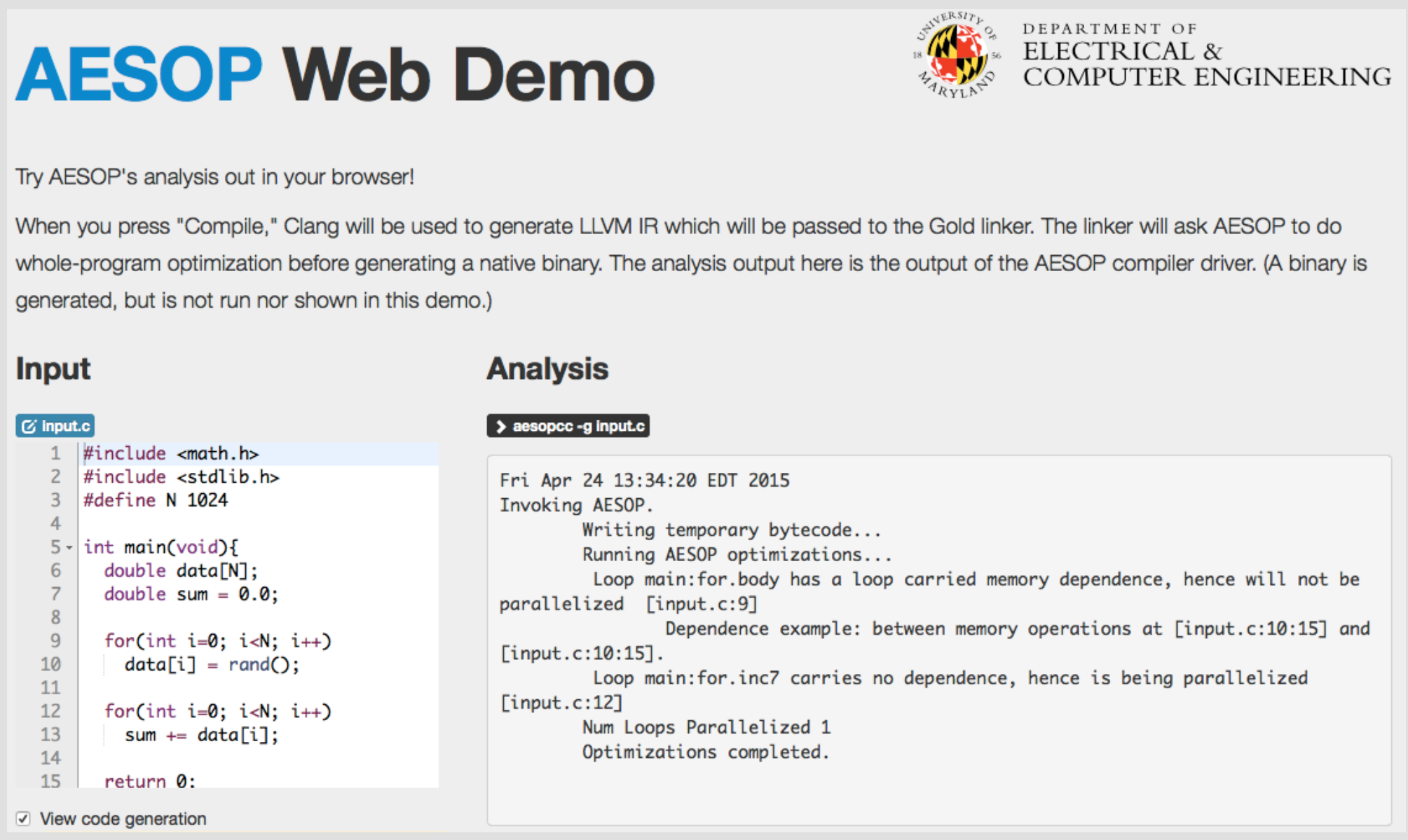


Source: https://wiki.smartos.org/display/DOC/SmartOS+Virtualization

- AESOP already open-source:
- SCAF soon to be open-sourced
- Investigating SCAF extensions for space-sharing VMs



Source: https://wiki.smartos.org/display/DOC/SmartOS+Virtualization



AESOP Web Demo

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

Try AESOP's analysis out in your browser!

When you press "Compile," Clang will be used to generate LLVM IR which will be passed to the Gold linker. The linker will ask AESOP to do whole-program optimization before generating a native binary. The analysis output here is the output of the AESOP compiler driver. (A binary is generated, but is not run nor shown in this demo.)

Input

input.c

```
#include <math.h>
#include <stdlib.h>
#define N 1024

int main(void){
  double data[N];
  double sum = 0.0;

  for(int i=0; i<N; i++)
    data[i] = rand();

  for(int i=0; i<N; i++)
    sum += data[i];

  return 0;
```

☑ View code generation

Analysis

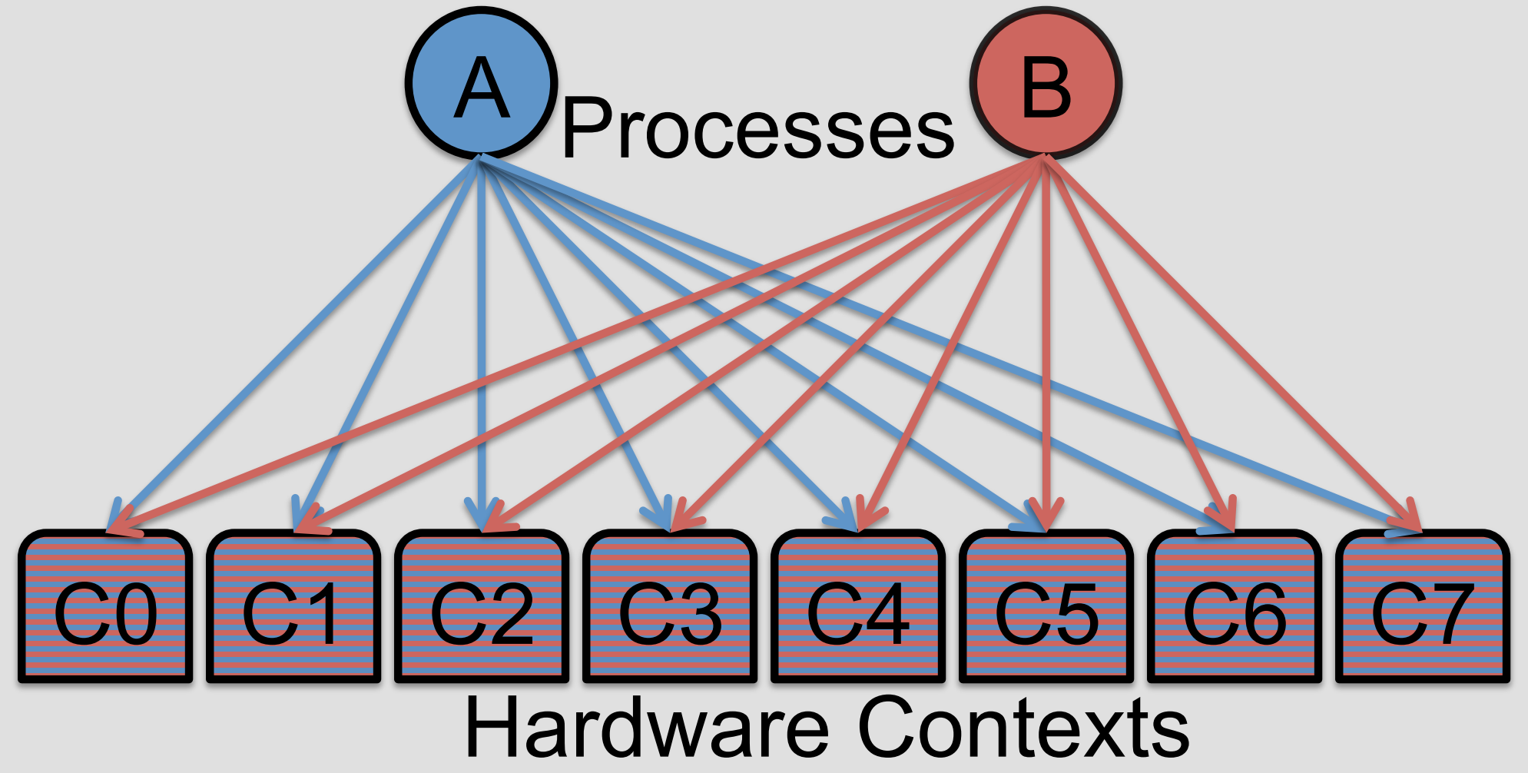
aesopcc -g input.c

```
Fri Apr 24 13:34:20 EDT 2015
Invoking AESOP.
        Writing temporary bytecode...
        Running AESOP optimizations...
         Loop main:for.body has a loop carried memory dependence, hence will not be
parallelized  [input.c:9]
                Dependence example: between memory operations at [input.c:10:15] and
[input.c:10:15].
         Loop main:for.inc7 carries no dependence, hence is being parallelized
[input.c:12]
        Num Loops Parallelized 1
        Optimizations completed.
```

http://aesop.ece.umd.edu

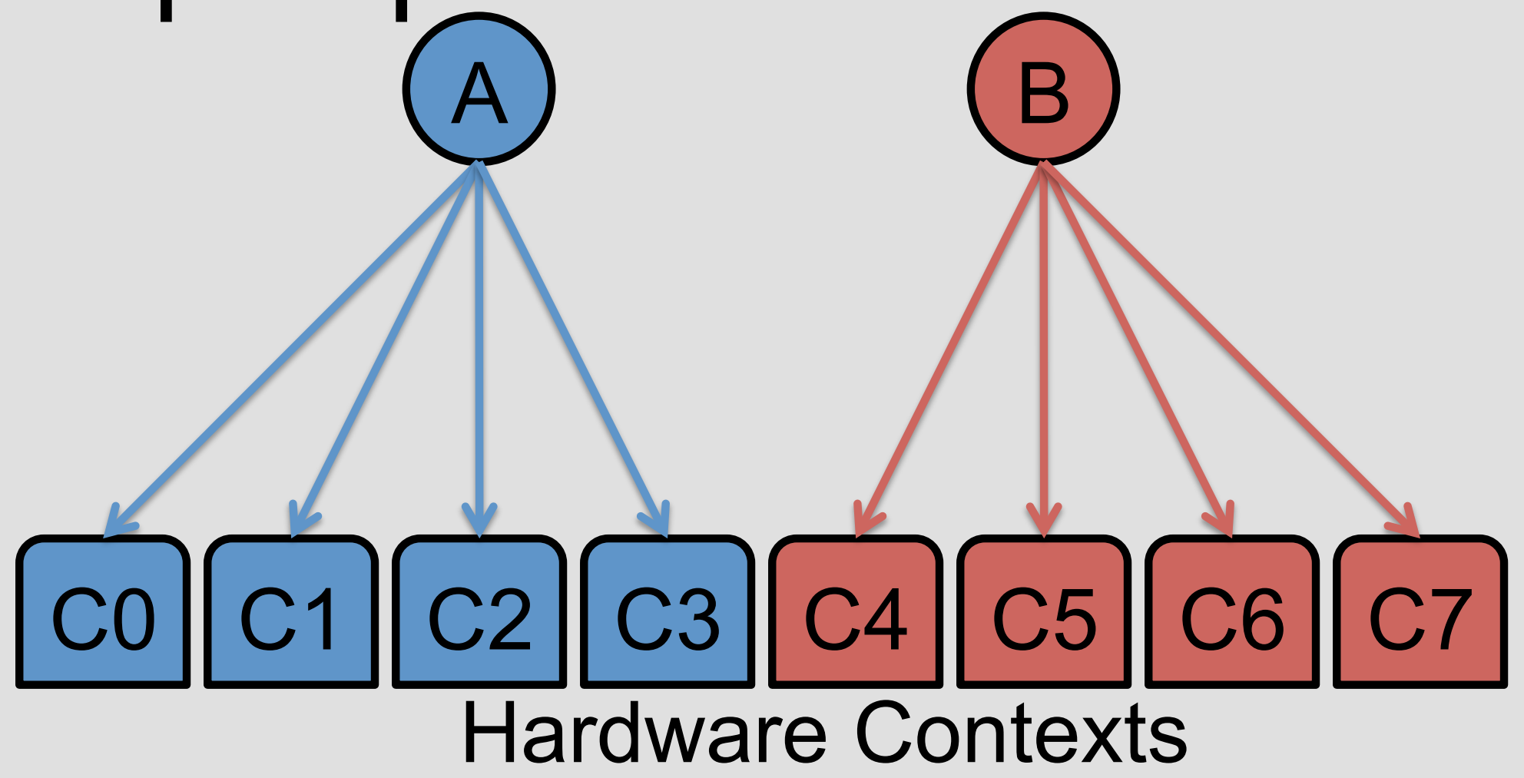
Previous systems:

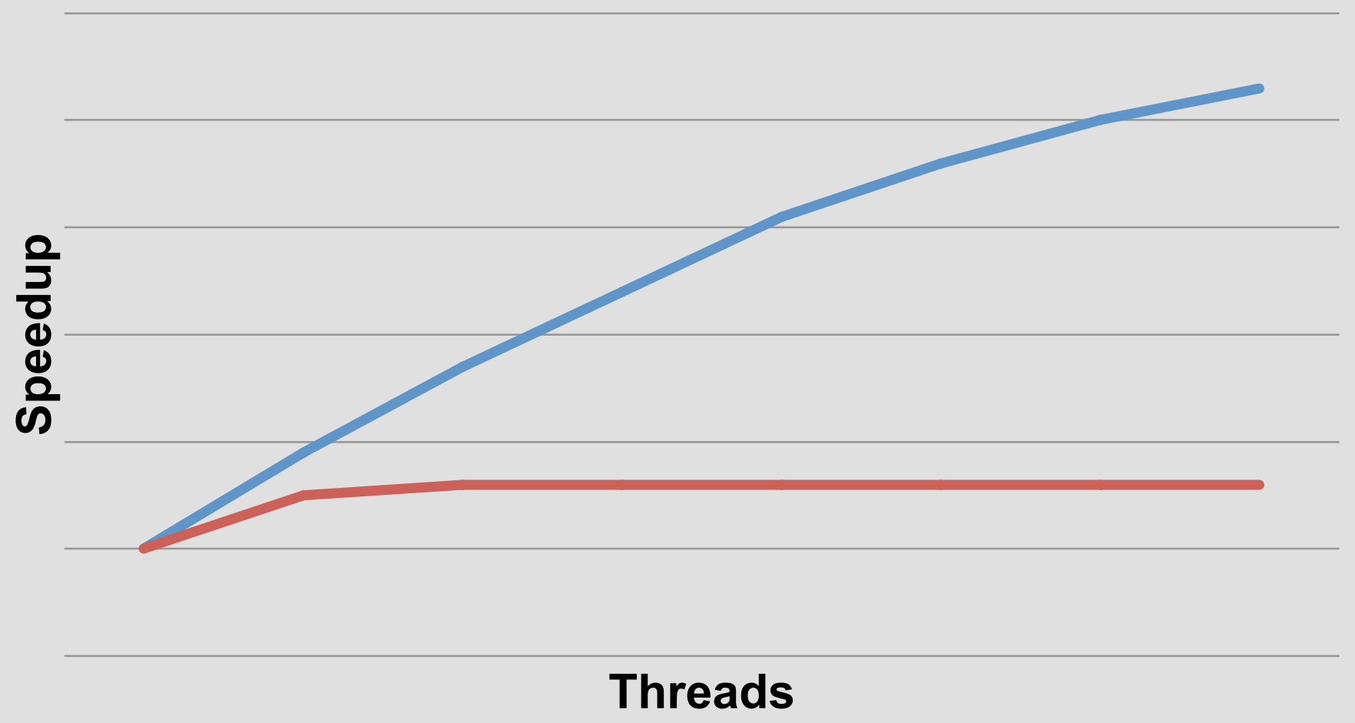
Unmodified systems: each process assumes it can occupy an entire physical machine



Result: oversubscription, context-switching, hardware contention

Equipartitioning: each process occupies an equal partition of the hardware contexts

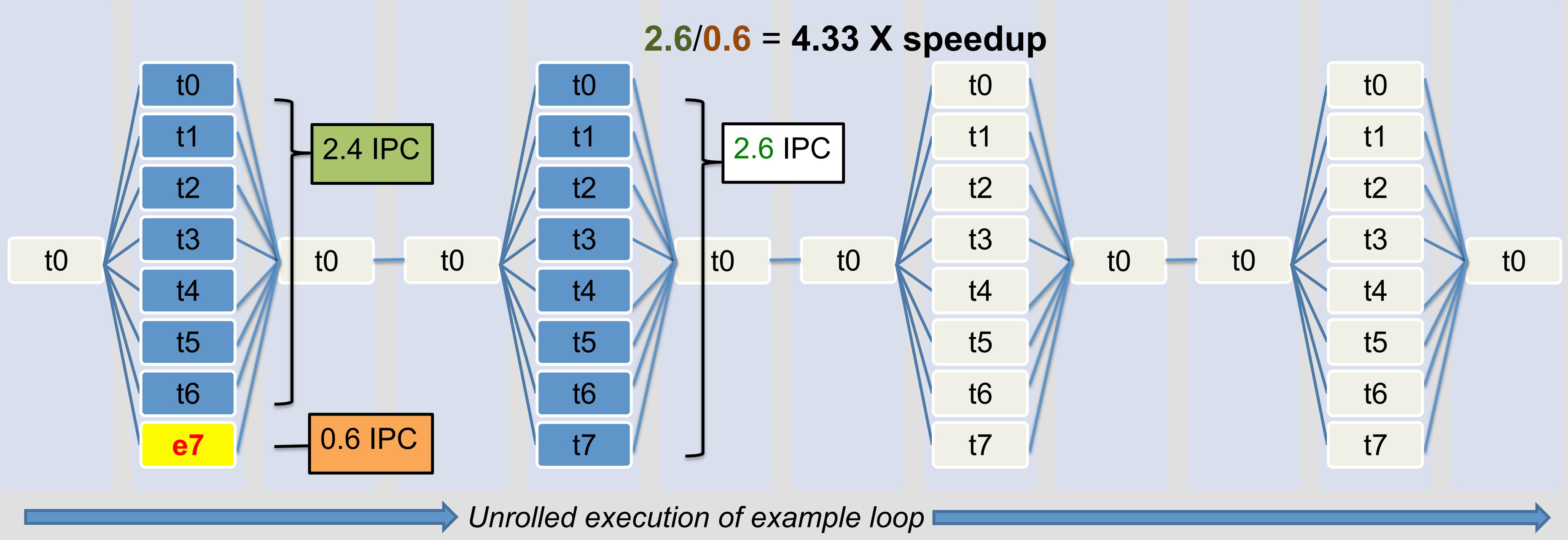




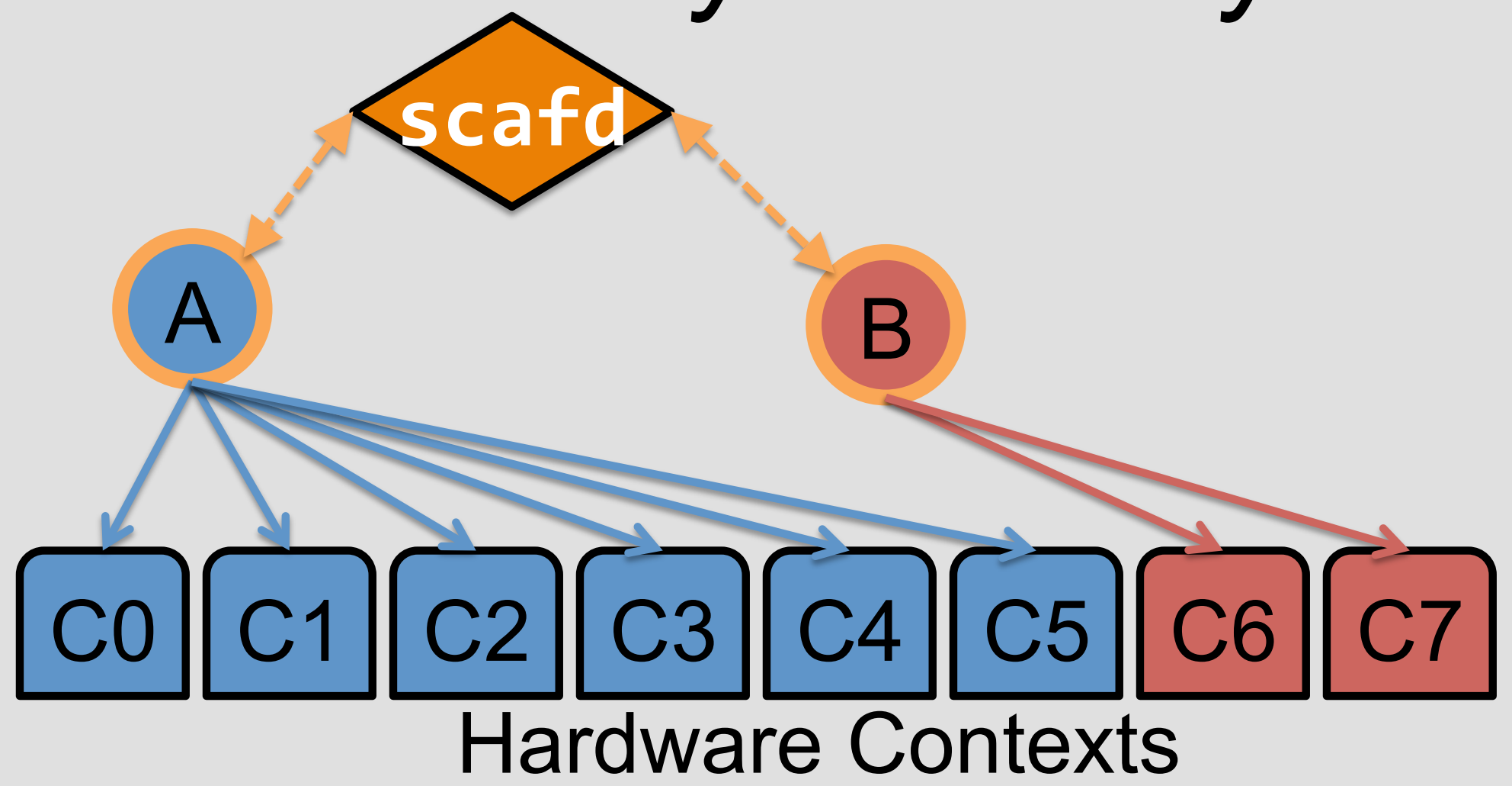
Result: dedicated HW contexts, no switching.
But what if $\text{Speedup}_A(4) >> \text{Speedup}_B(4)$?

What can we do?

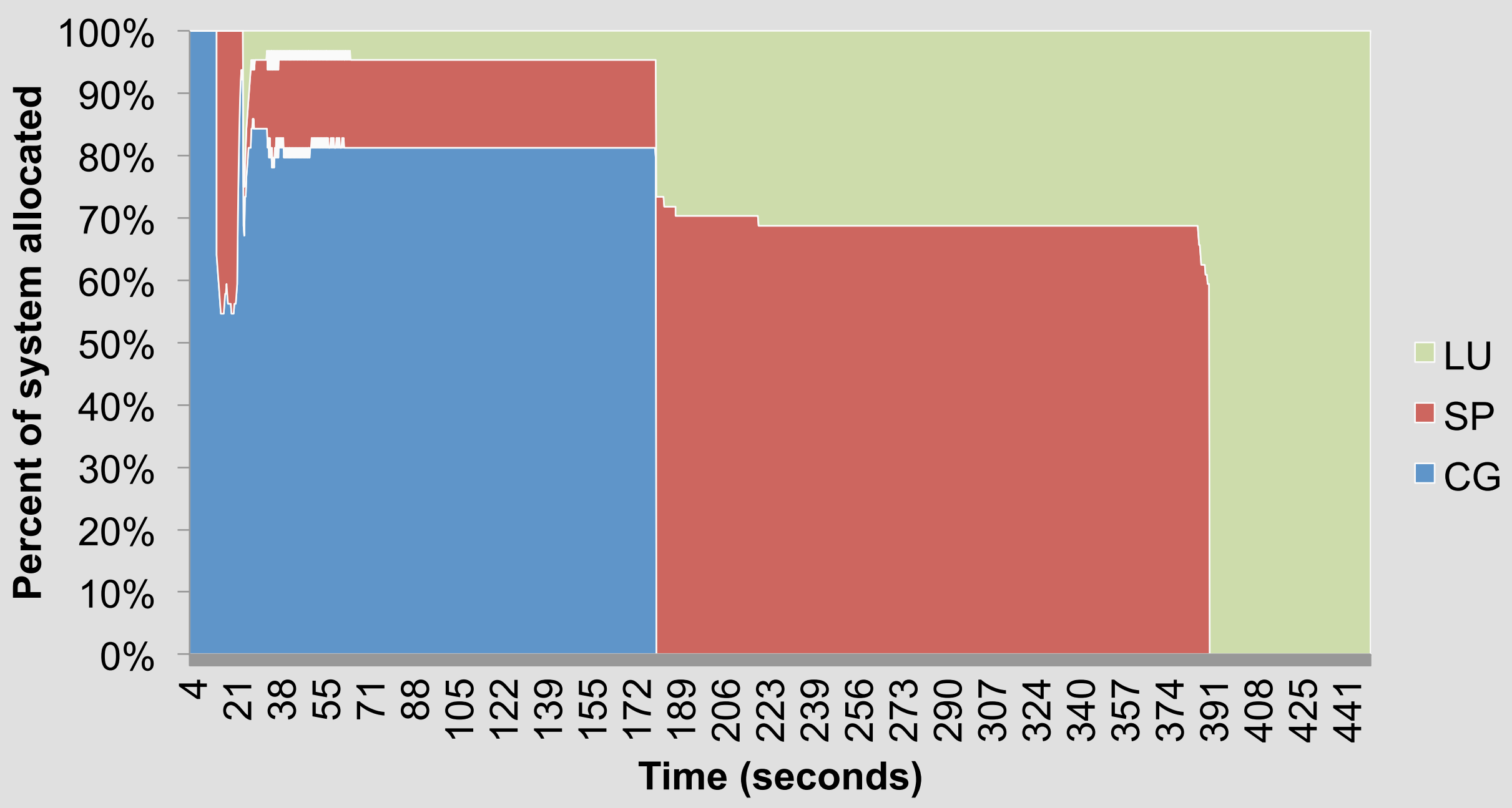
SCAF: Perform *on-line experiments* to reason about speedups



Coordinate an improved allocation *dynamically*



Example: 3-way SCAF multiprogramming



Summary of results:
Mean improvement of 1.11-1.22X sum-speedup for 80-89% of pairs vs. EQ; 1.27-1.7X for 72-100% of pairs vs. unmod on 4 platforms

This work was supported by a NASA Office of the Chief Technologist's Space Technology Research Fellowship, NASA Grant #NNX11AM99H

Systems and Computer Architecture Lab