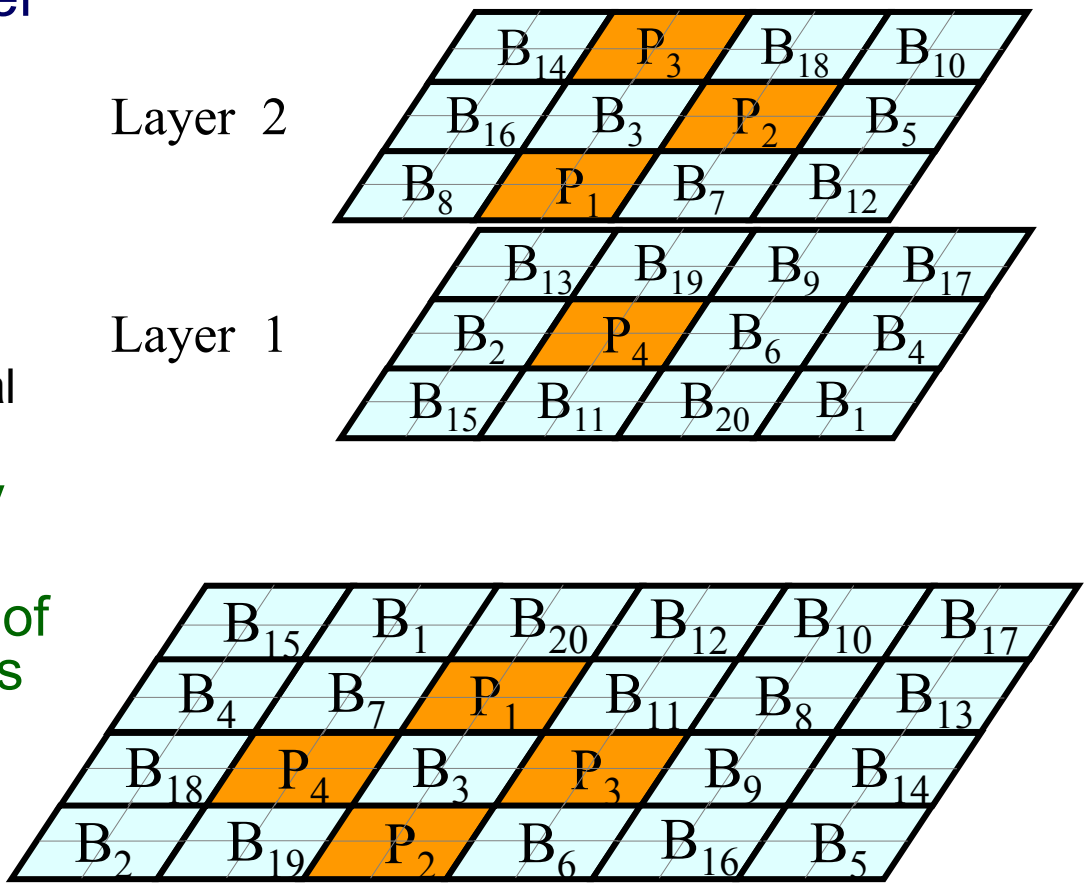
# CS 423

## Current Directions

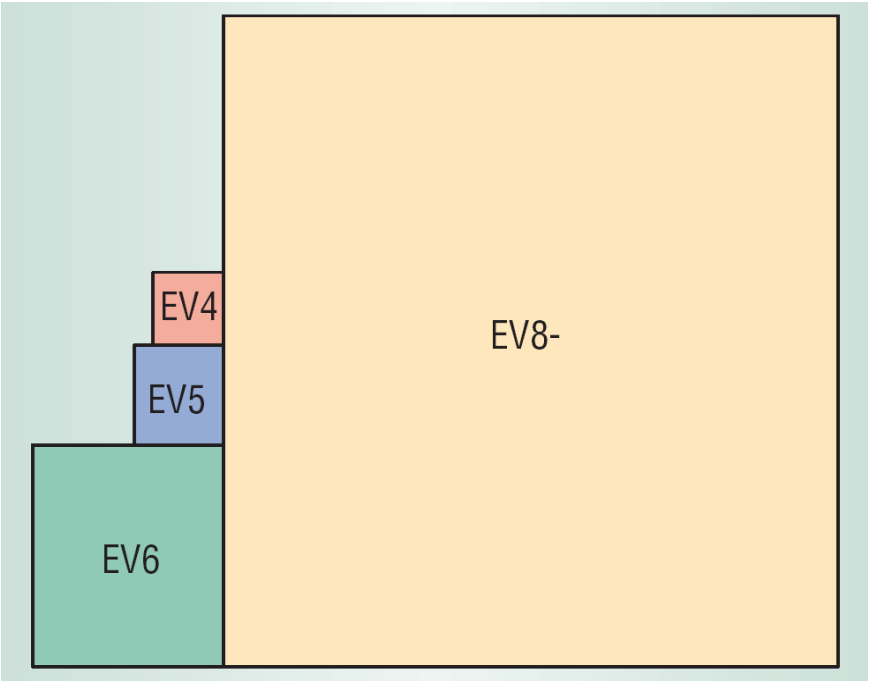
# Memory Issues in 3D Architectures

- Benefits of a 3D chip over a 2D design
  - Reduction on global interconnect
    - Performance: reduced average interconnect length
    - Power: reduction in total wiring length
  - Higher packing density and smaller footprint
  - Support for realization of mixed-technology chips



# Heterogeneous Chip Multiprocessors

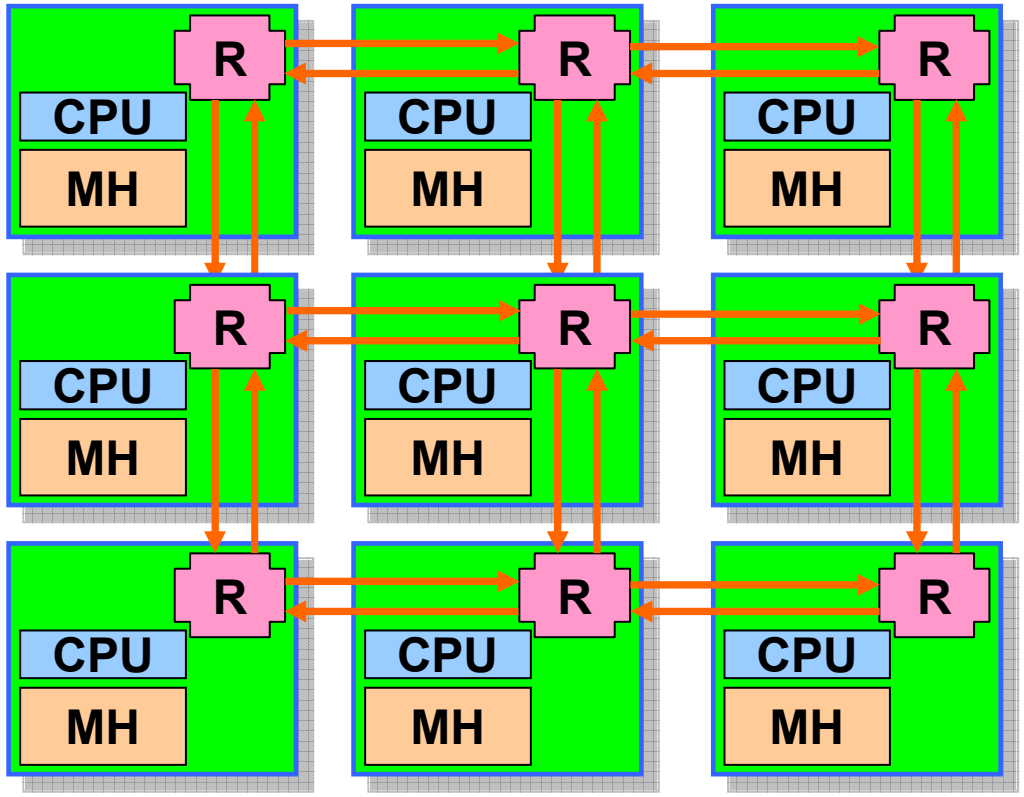
- A chip multiprocessor
  - High-complexity cores
  - Low-complexity cores
- Better resource-to-application mapping
  - Speed of a large core
  - Efficiency of a small core




- Alpha Cores
- EV8 is 80X bigger
- Only 2X – 3X performance improvement

# NoC Architecture

- M × N mesh architecture
- Node in the mesh
  - Processor
  - Memory module
  - Switch



# Background

- Intel Many Integrated Core (Intel MIC) architecture
  - Processing highly parallel workloads
  - Standard programming models (OpenMP and Cilk with a few extensions)

**Knights Ferry**

Packaged as a co-processor in a PCI-e card. With 32 cores running four threads apiece, this can process 128 threads at 1.2GHz.

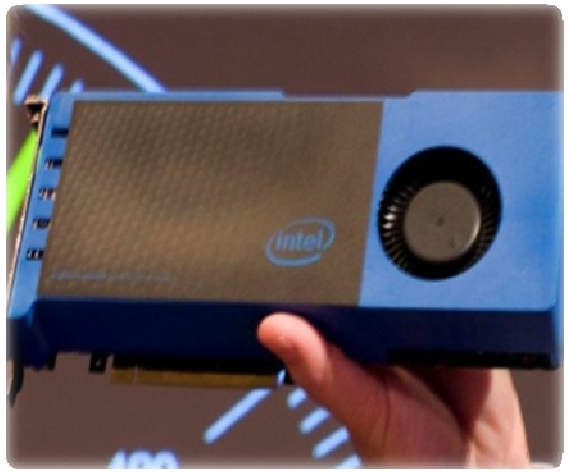
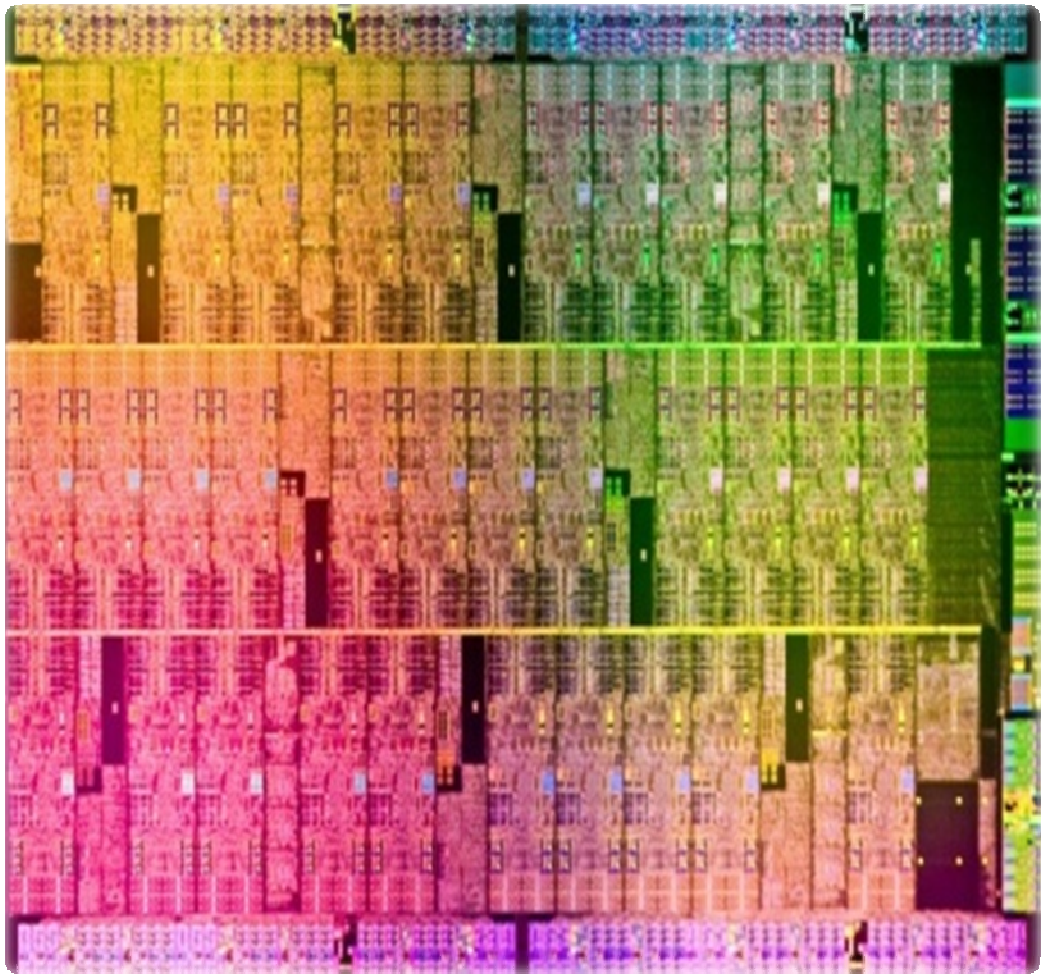



Photo source: ZDNet

# Background



## Problems with current parallelization techniques

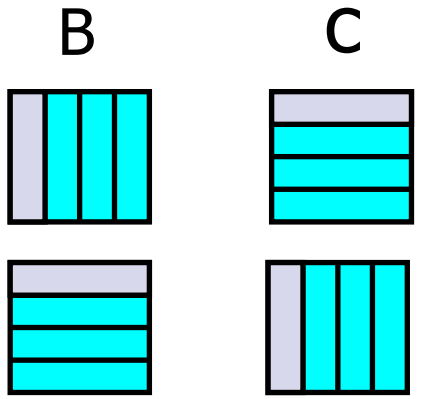
- Developed in context of high performance parallel machines
- Most of them parallelize one loop nest at a time
  - Cannot capture inter-nest relations well
- Their main goal is to minimize inter-processor communication
  - Not very suitable for chip multiprocessors
- What we need is data reuse oriented whole program parallelization

# Example

```
for(i=1;i≤n; i++)
  for(j=1;j ≤n; j++)
    A[i][j] += B[j][i]+C[i][j]
```

i loop is parallelized

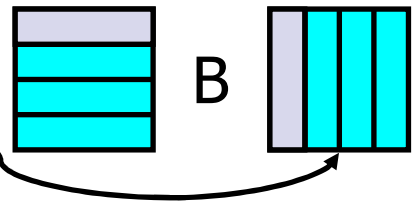
j loop is parallelized




```
for(i=1;i≤n; i++)
  for(j=1;j ≤n; j++)
    D[i][j] = D[i][j]+B[i][j]
```

i loop is parallelized

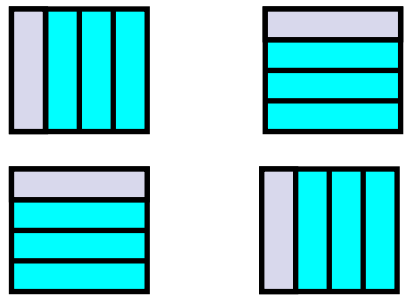
j loop is parallelized




```
for(i=1;i≤n; i++)
  for(j=1;j ≤n; j++)
    B[j][i] = B[j][i]+C[i][j]
```

i loop is parallelized

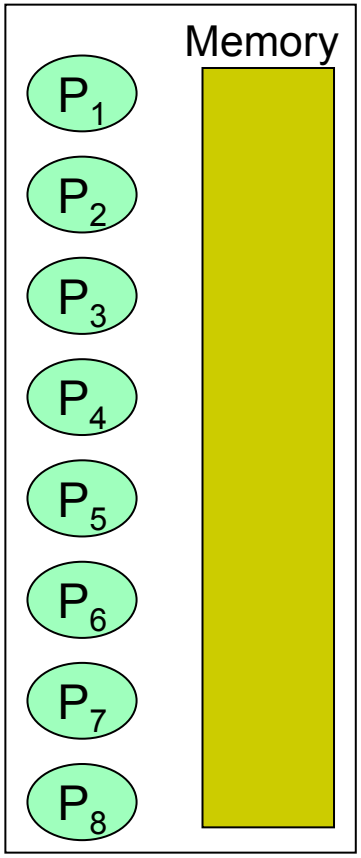
j loop is parallelized



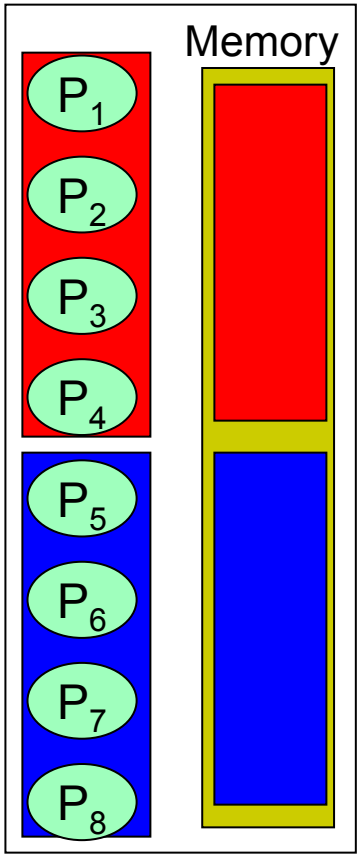
# Resource Allocation

- Prior OS-based resource partition approaches
  - Advantage: transparent to applications and programmers
  - Drawback: application oblivious and reactive
- Our goal : *Proactive* resource partitioning scheme

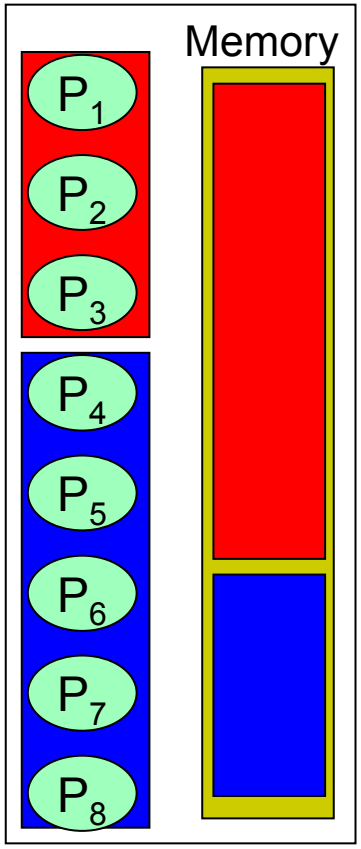
# Architecture and Resource Partition Schemes



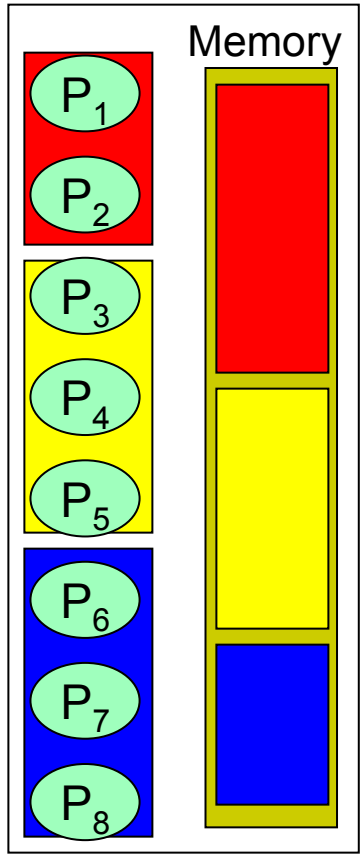

(a) Abstract view of an MPSoC architecture with 8 processors




(b) Equal partitioning of resource across two applications

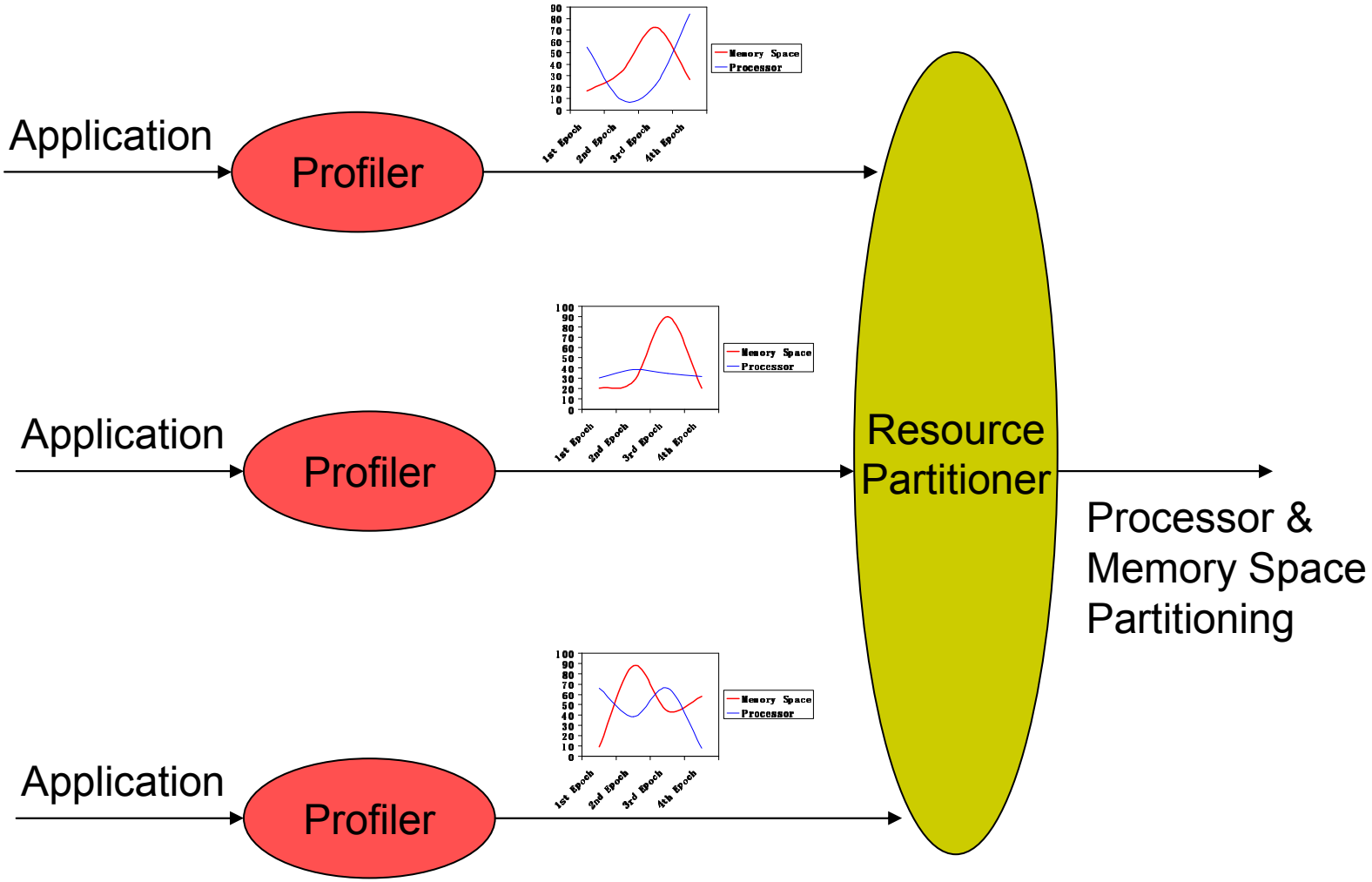



(c) Nonuniform partitioning of resources across two applications

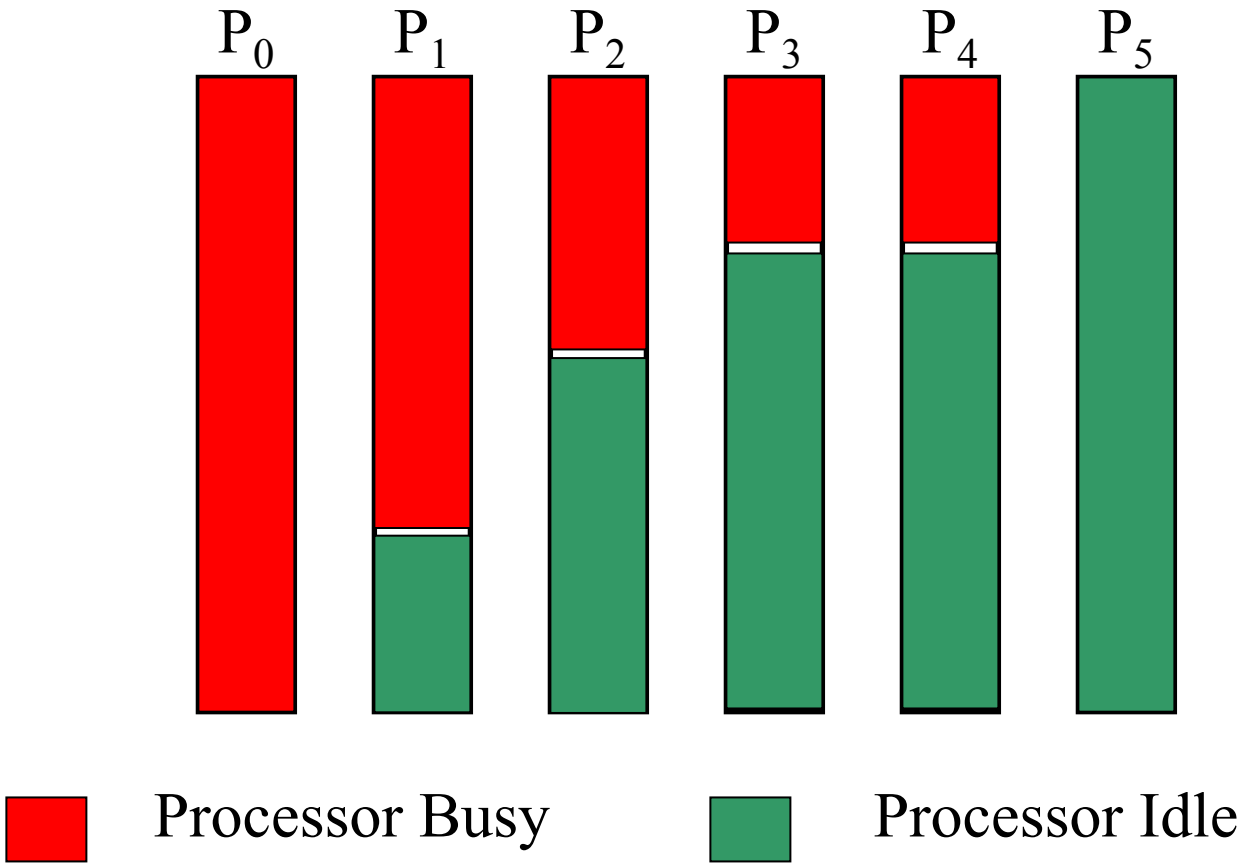



(d) Nonuniform partitioning of resources across three applications

# Components of Our Approach



# Scenario with no energy saving scheme



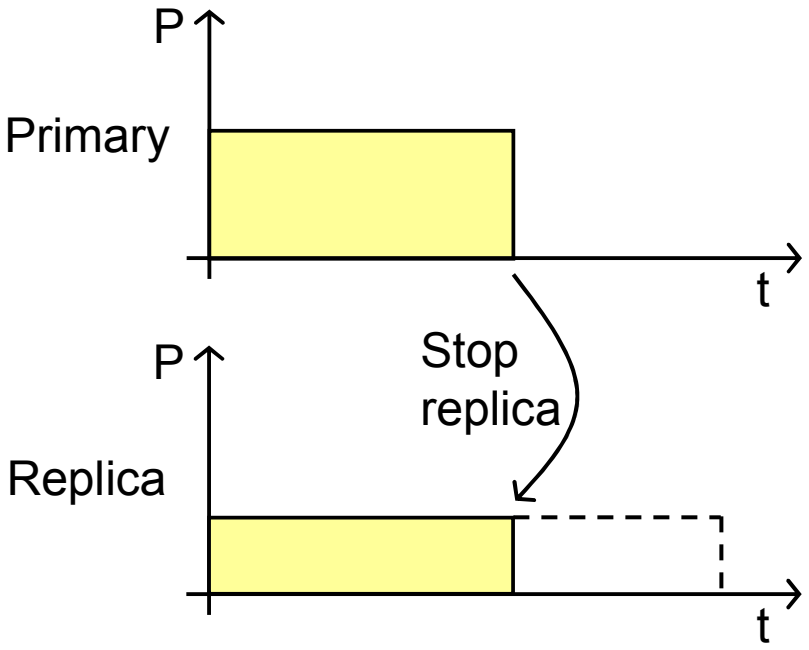
# Energy Reduction Schemes

- There are two primary groups
  - Voltage scaling techniques
  - Processor shutdown schemes
- They can be applied using hardware or an optimizing compiler
- They are applied independently
- They are applied in disjoint manner

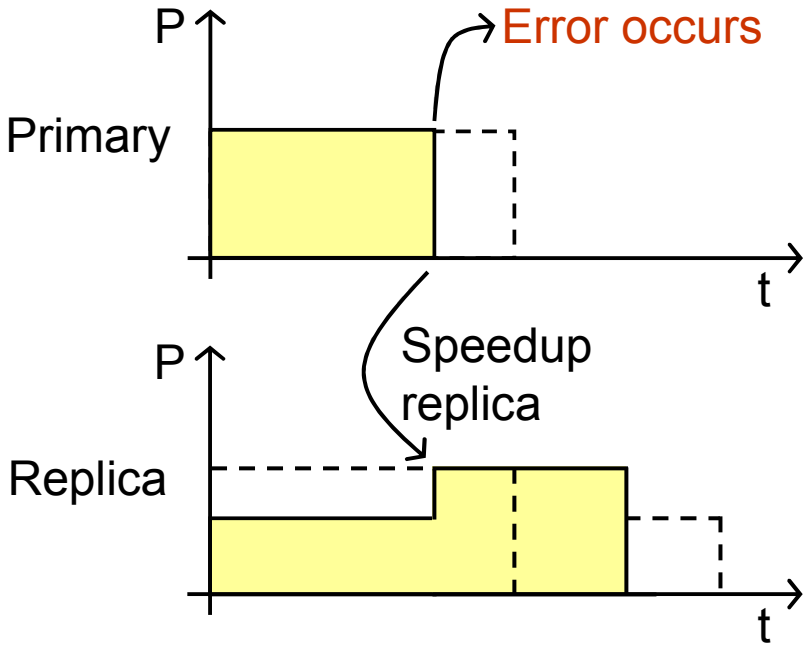
# Our Approach

- When no errors
  - Determined by the successful termination of the primary copy
  - Terminate the replica
  - Since the replica has operated with lower voltage/frequency so far, we save energy, compared to the case where the replica is executing with the highest voltage/frequency available
- When an error occurs in the primary copy
  - Primary copy is aborted
  - Replica is switched to the highest voltage/frequency level to minimize the time to complete the task

# Our Approach




Error Free

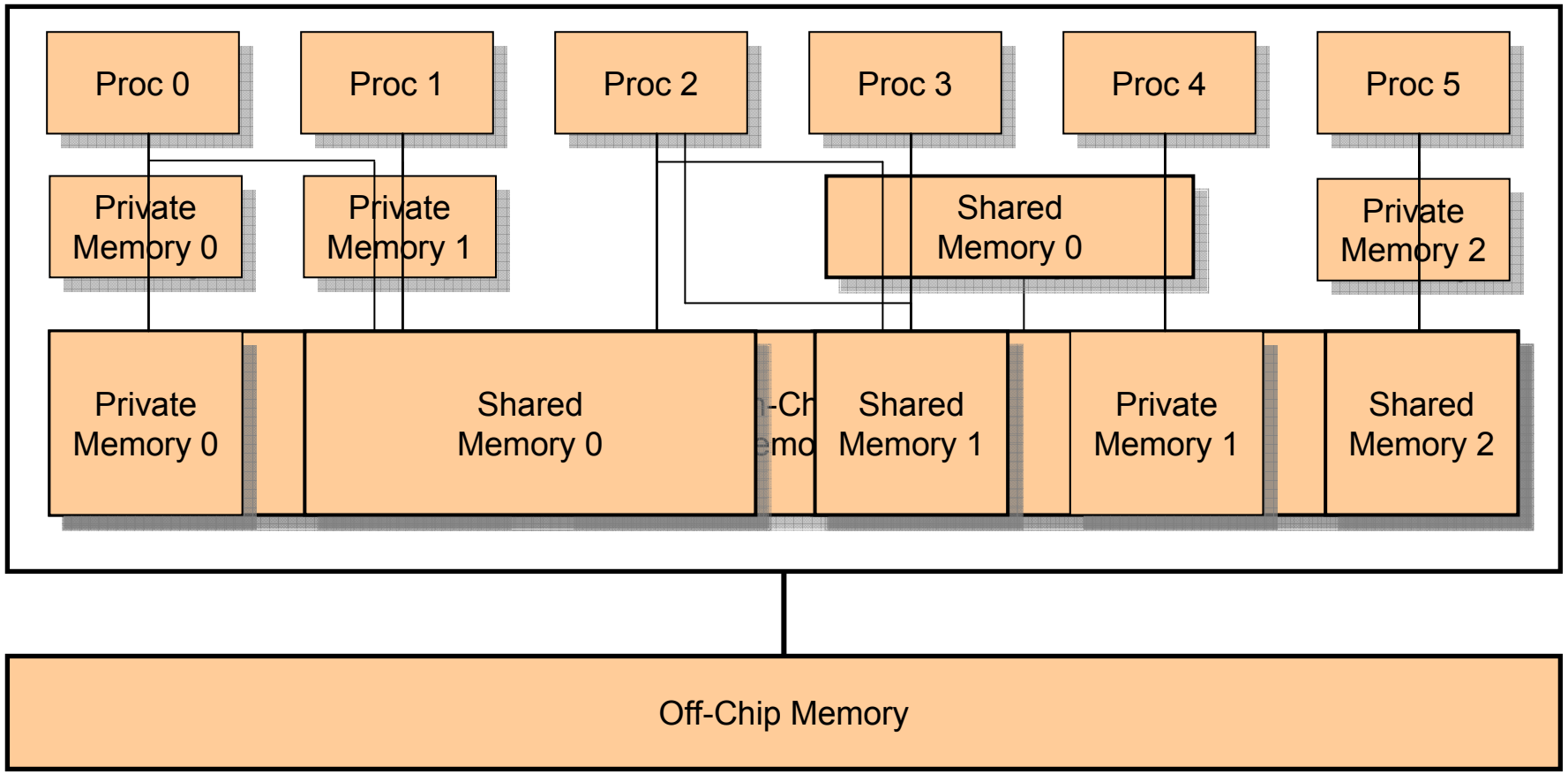



Transient Error

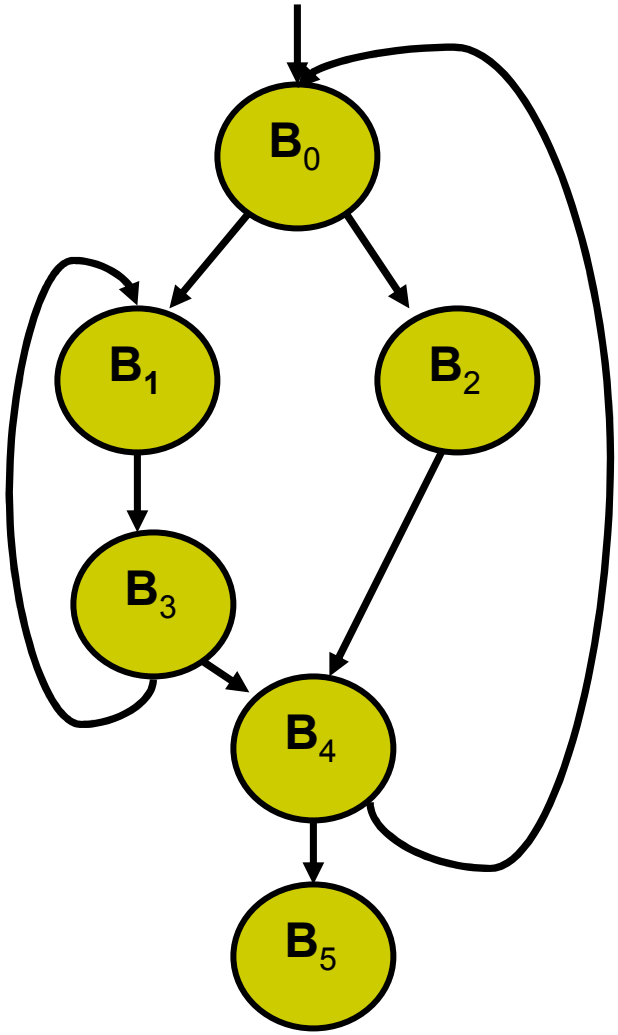
# Memory Hierarchy Design (2/5)

- Memory hierarchy management has been well studied
  - Caches
  - From the performance perspective
- Relatively less attention
  - Software managed memories
  - Optimizing energy behavior
- Software-managed hierarchies can be preferable against hardware counterparts
  - Able to design a customized memory hierarchy that suits the needs of the application
  - Data flow is managed by software
    - Energy-efficient → Dynamic lookup in hardware

# Memory Hierarchy Design (4/5)



# Program Representation




- Our approach works on a control flow graph (CFG).
  - Nodes: Basic blocks
  - Edges: Control flow (conservative)