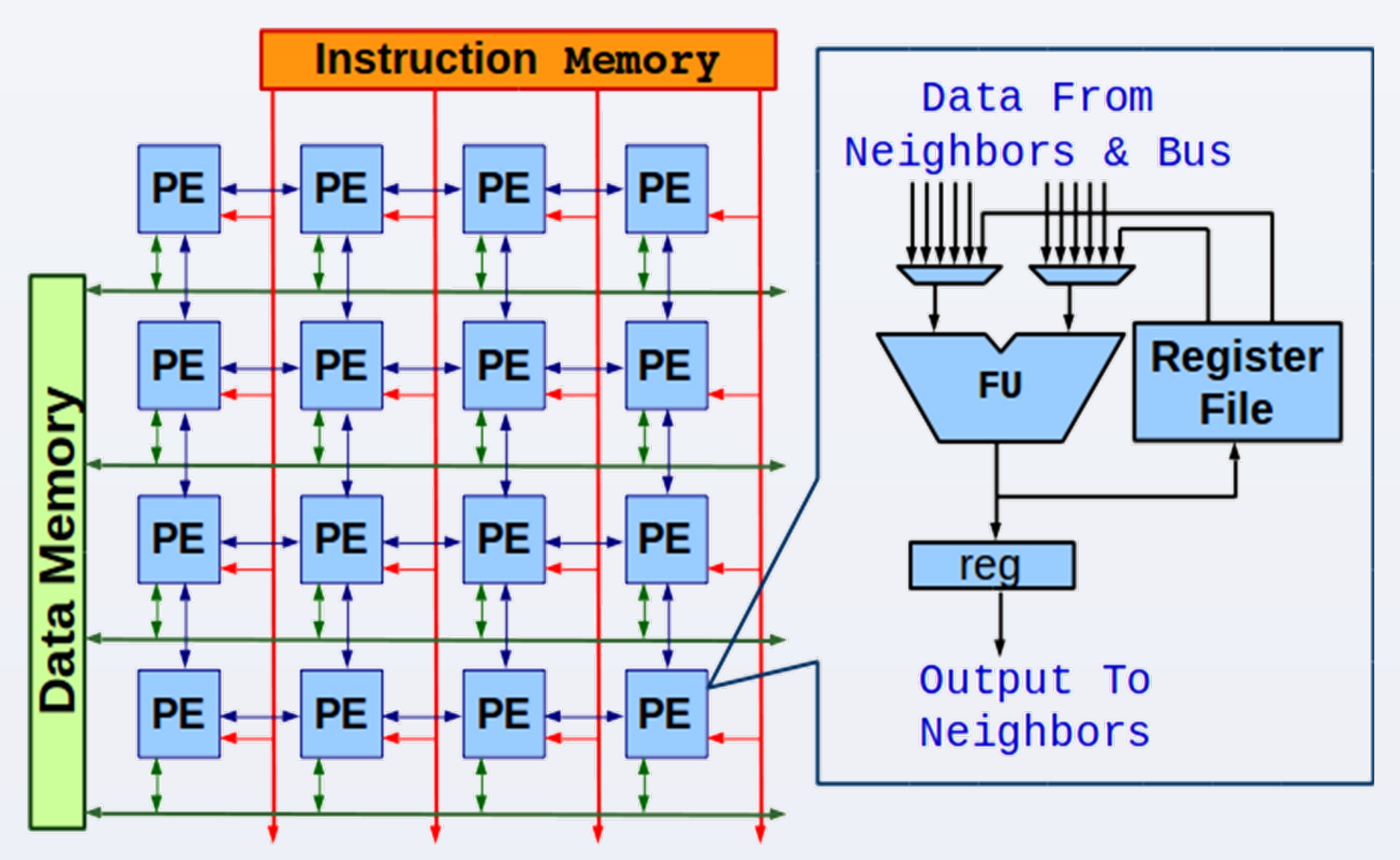
RAMP: Resource-Aware Mapping for CGRAs

Shail Dave, Mahesh Balasubramanian, Aviral Shrivastava

Compiler Microarchitecture Lab, Arizona State University

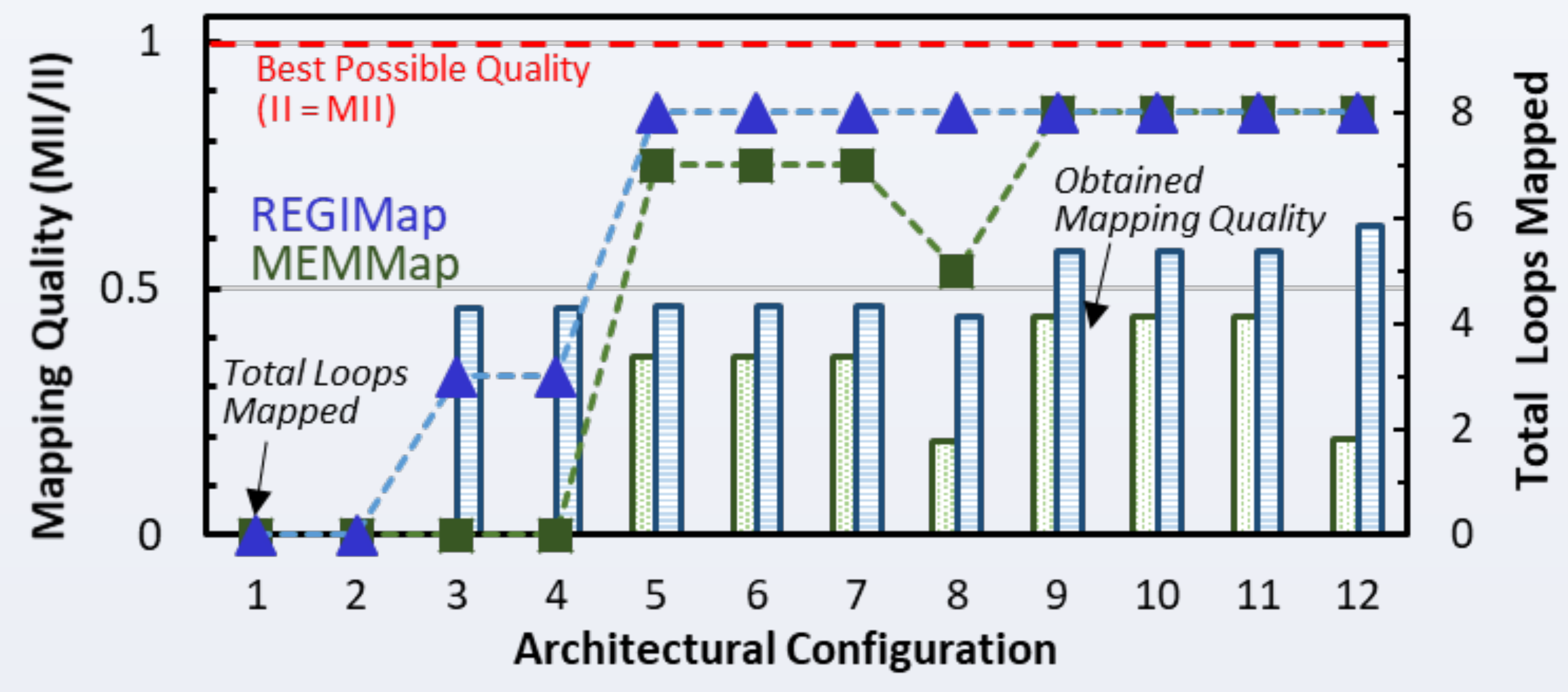
Coarse Grained Reconfigurable Arrays (CGRAs)

- Array of Processing Elements (PEs); each PE has ALU-like functional unit that works on an operation every cycle.
- Power-efficiency of several 10s of GOps/Sec per Watt!
  - ➢ ADRES [HiPEAC '08]
  - ➢ HyCUBE [DAC '17]



A 4x4 CGRA with Local Register File and 2-D Mesh Interconnect

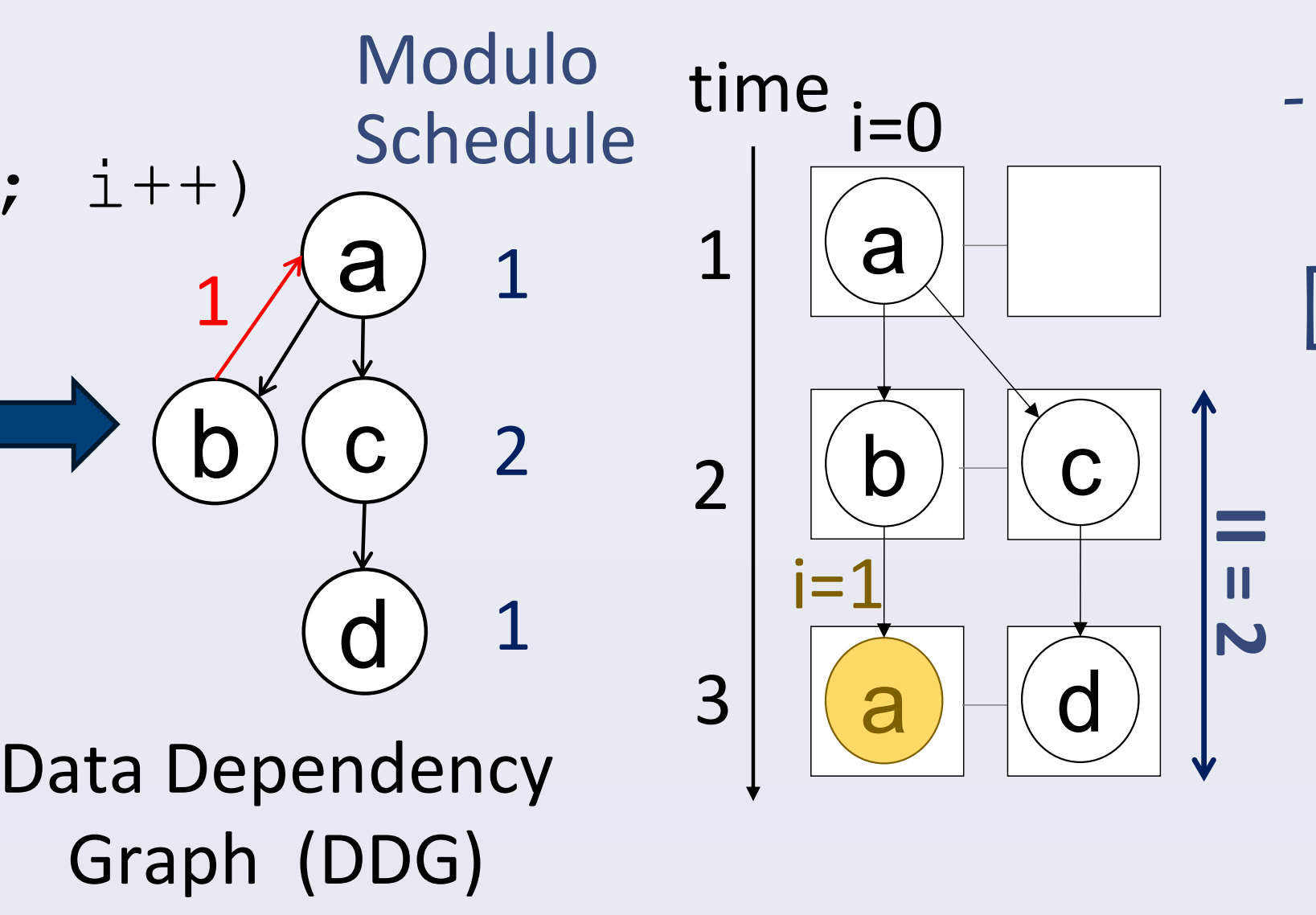
Performance Impact of Ad-Hoc Routing Strategies



Mapping Loops on CGRAs

```
B = 0;
for(i=0; i<1000; i++)
{
a:  A = B - 4;
b:  B = A + L;
c:  C = A * 3
d:  D = C + 7;
}
```

Sample Loop

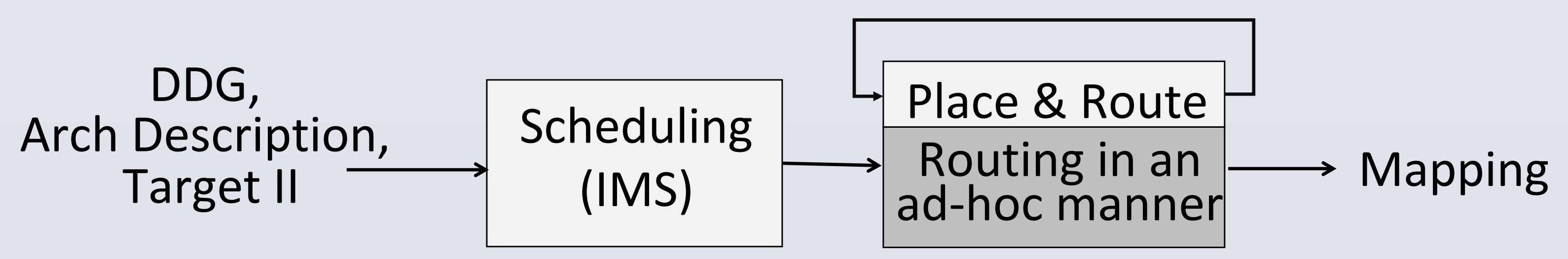


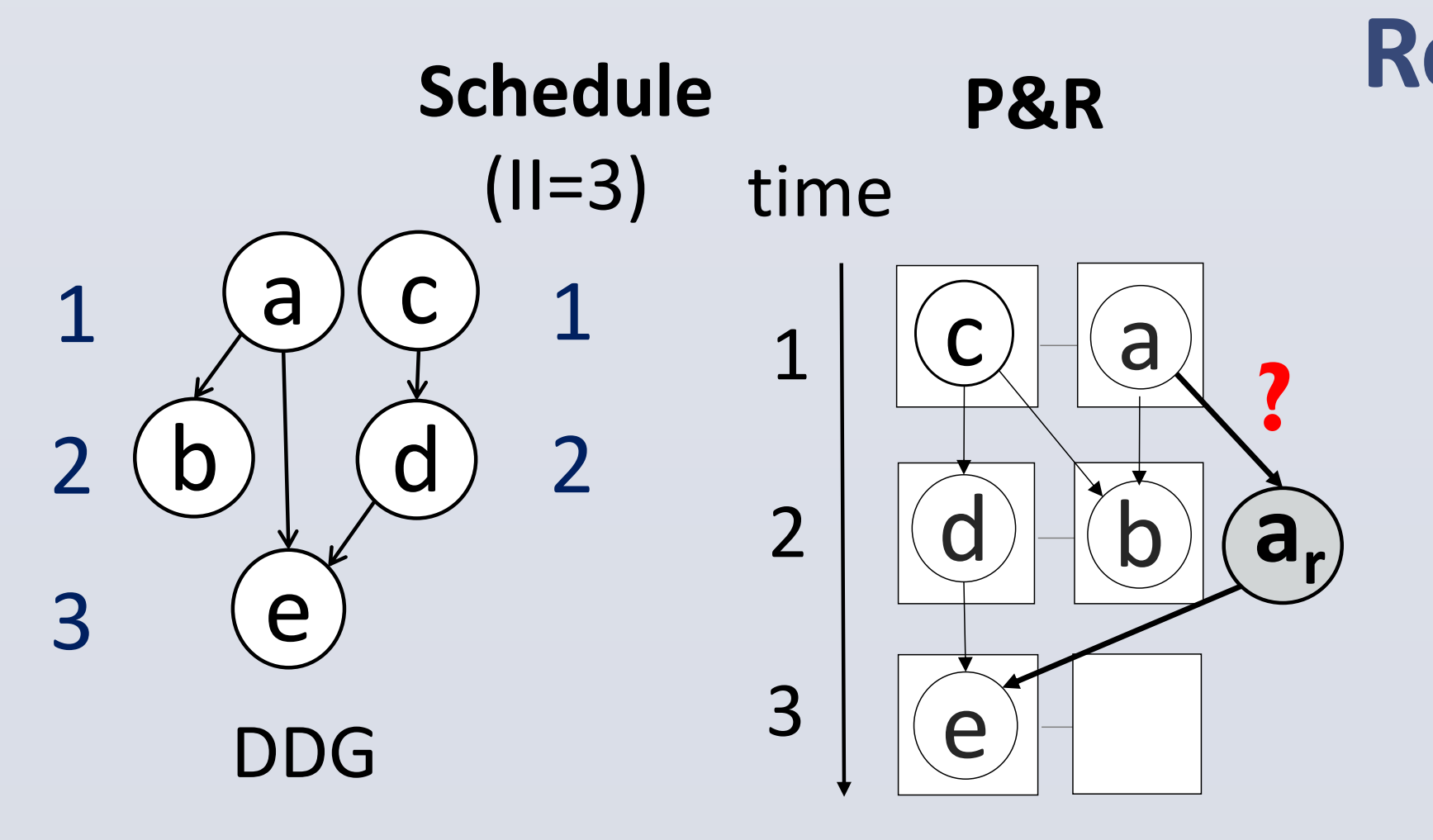
Data Dependency Graph (DDG)

- Iterative Modulo Scheduling [B. Rau, MICRO '94]
  - Software Pipelined Execution

- Performance Critically Depends on the Obtained Mapping
- **Mapping Problem = Routing Problem**
  - ➢ Routing is needed when the dependent operations are scheduled at a distant time, or operations cannot be mapped due to resource constraints.

Challenges with Code Generation Heuristics Employing Ad-hoc Routing Strategies





Routing Data Dependency via PEs

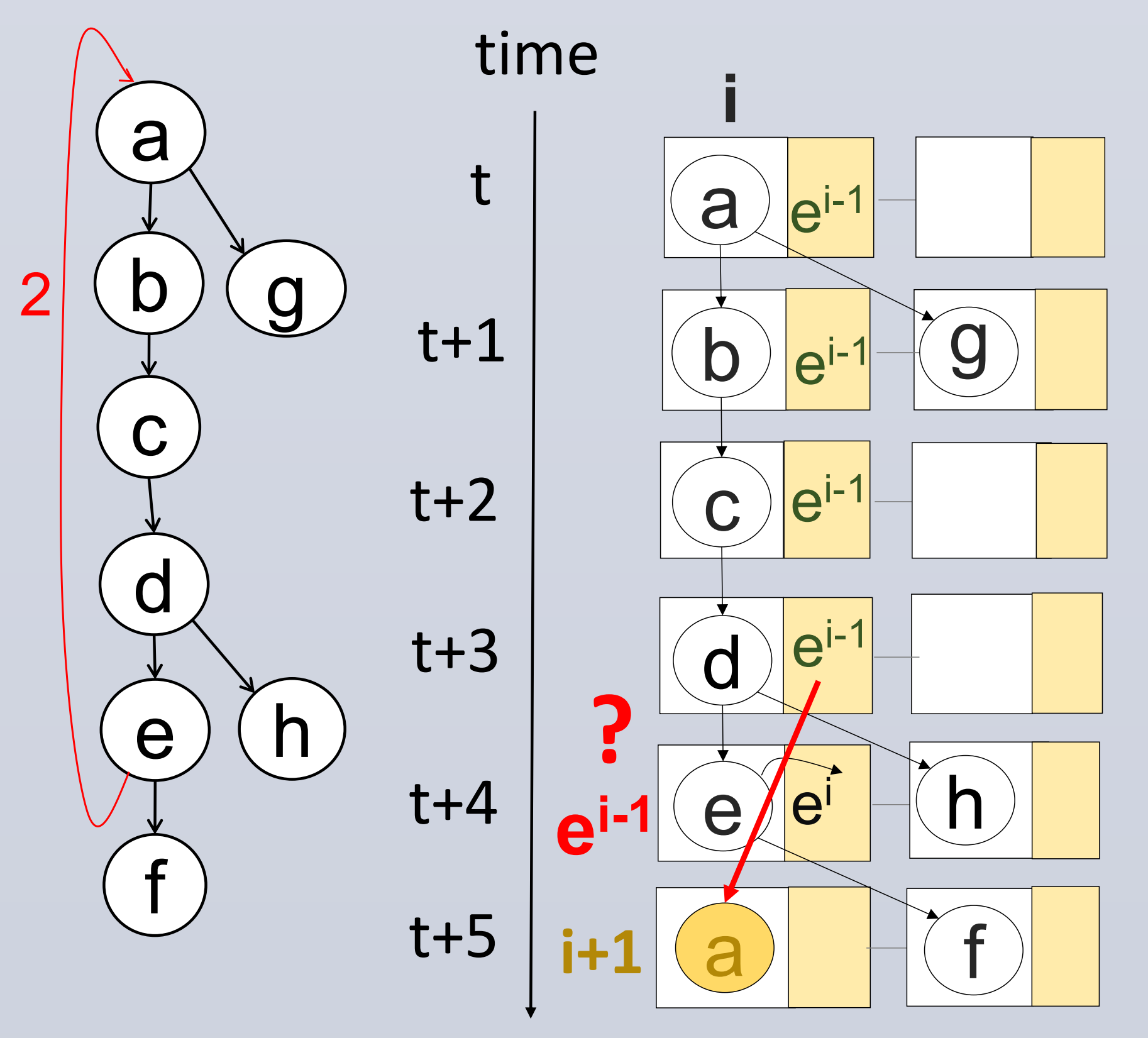
- **EMS** [H. Park et al., PACT '08]
- **EPIMap** [M. Hamzeh et al., DAC '12]

Since a_r is not rescheduled, cannot route a → e

Routing via Registers

- **REGIMap** [M. Hamzeh al.,DAC '13]
- **GraphMinor** [L. Chen et al., TRETS '14]

Cannot efficiently utilize distributed RFs to route $e^{i-2} \rightarrow a^i$

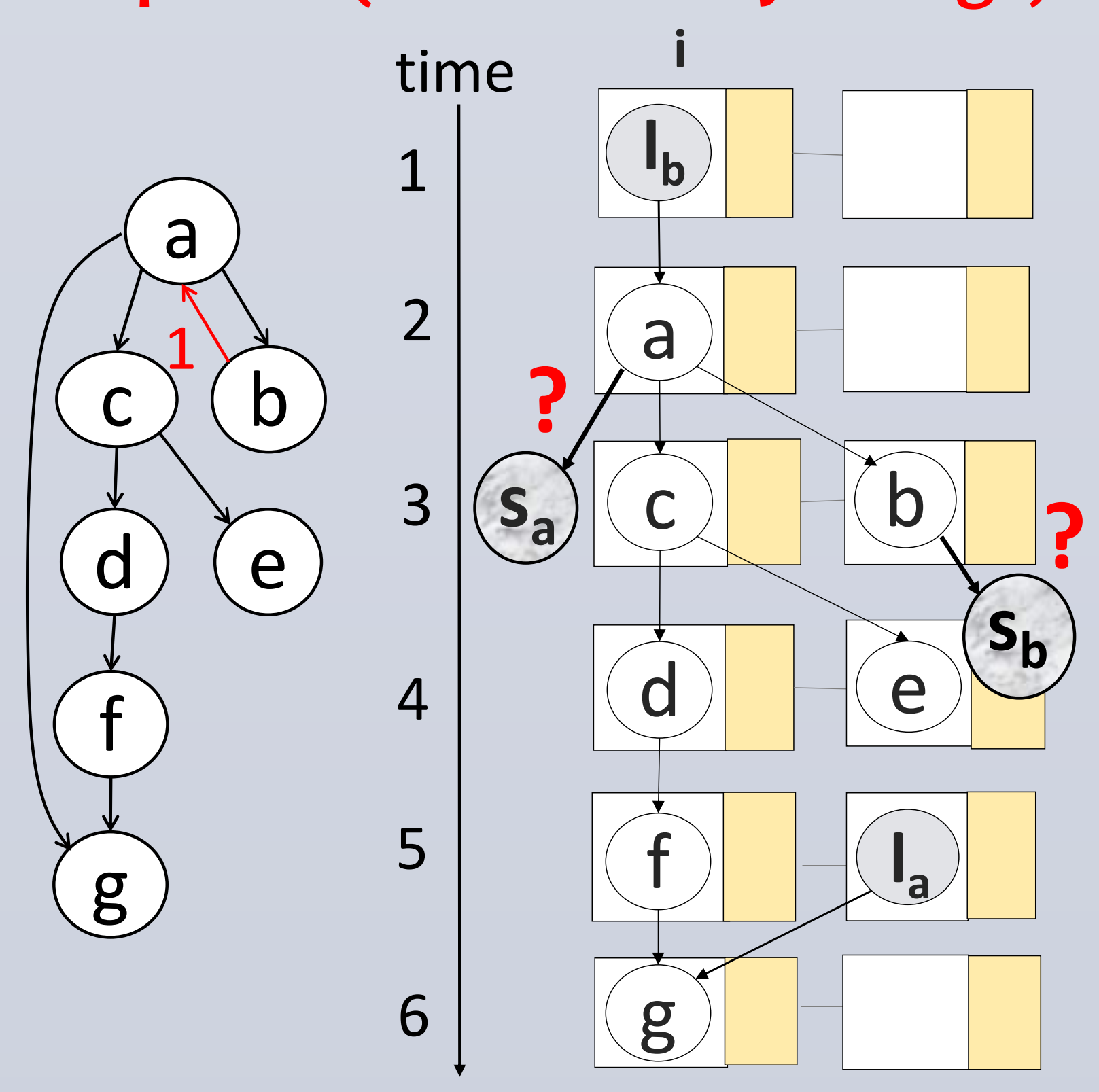


Routing via Memory

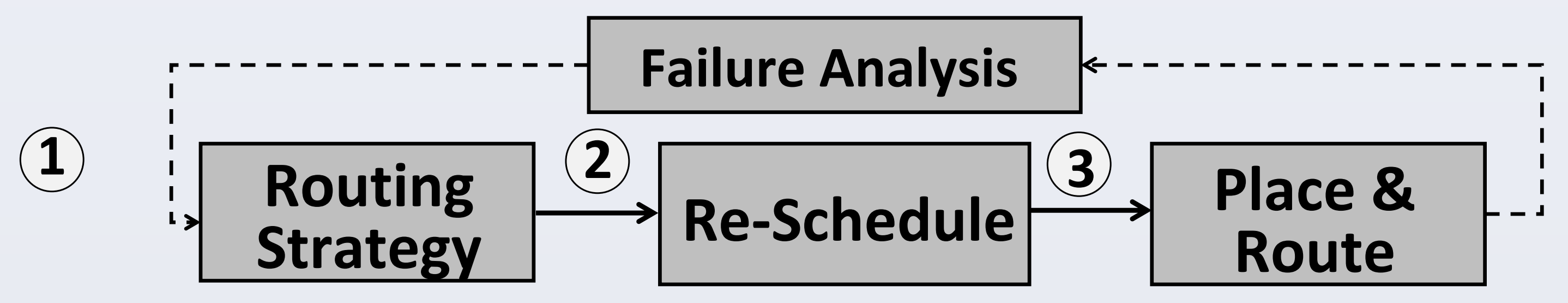
- **MEMMap** [S. Yin al., TVLSI '16]

Statically determine the dependencies routed via memory

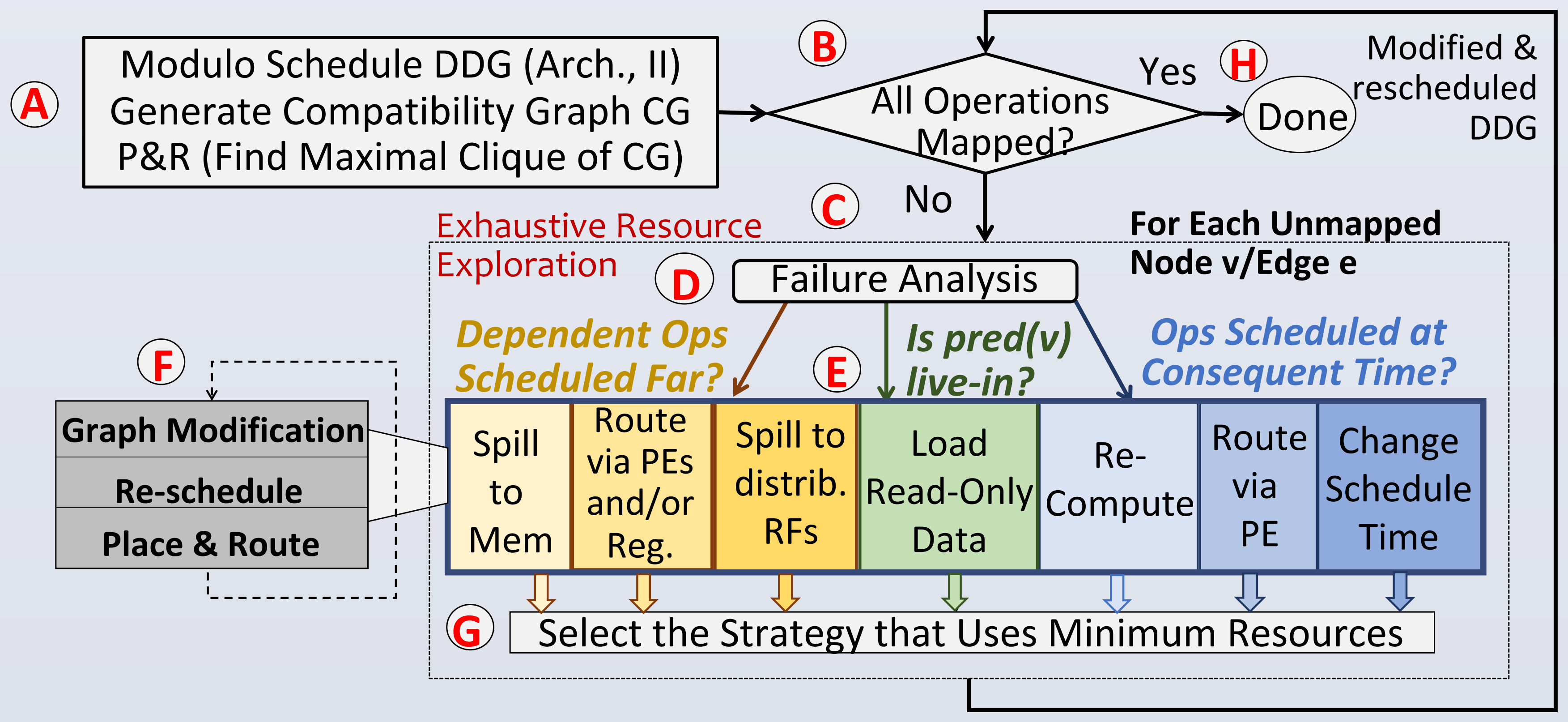
≠ spill data to memory when required (unavailability of regs)



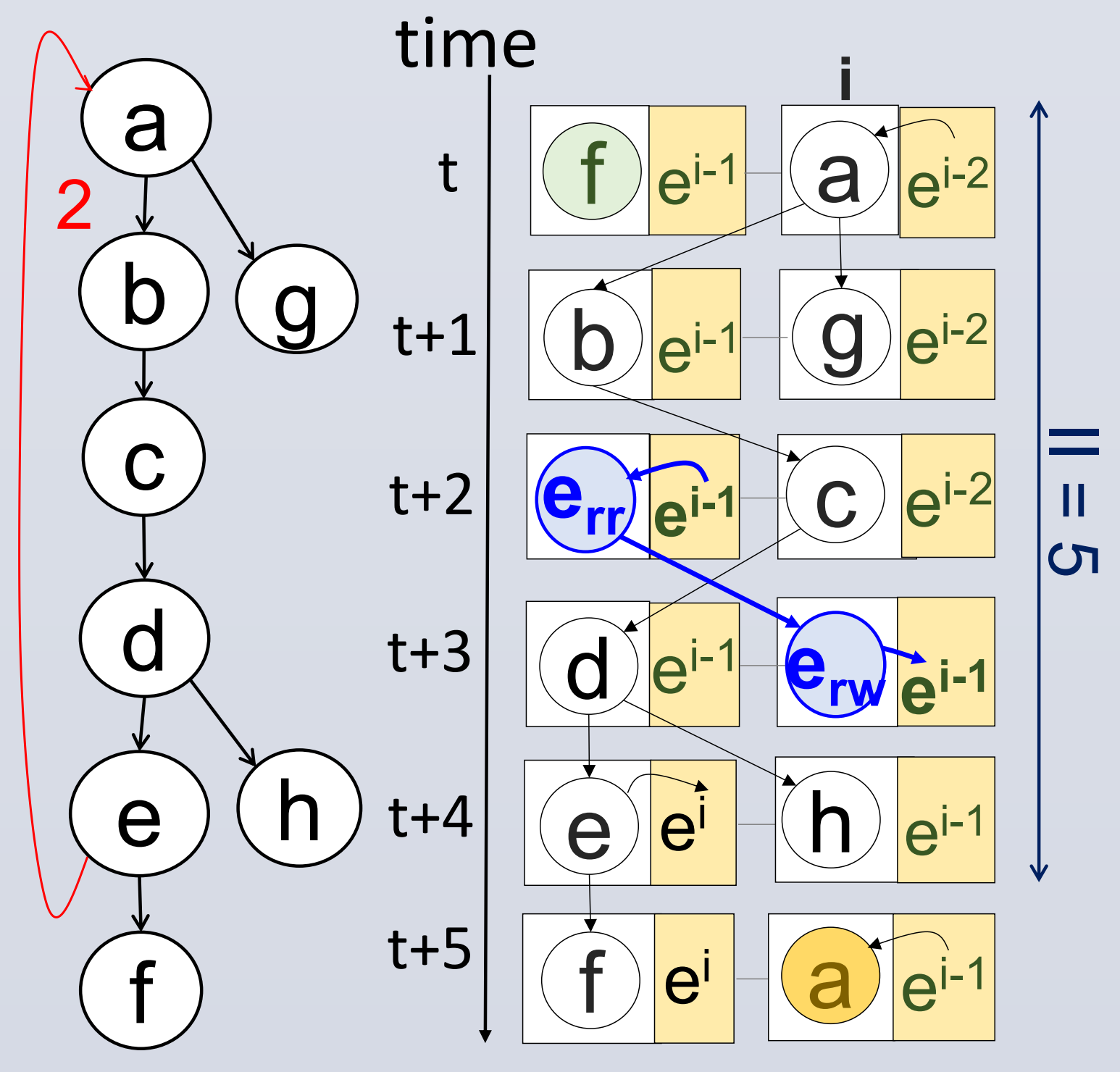
RAMP: Resource-Aware Mapping Technique



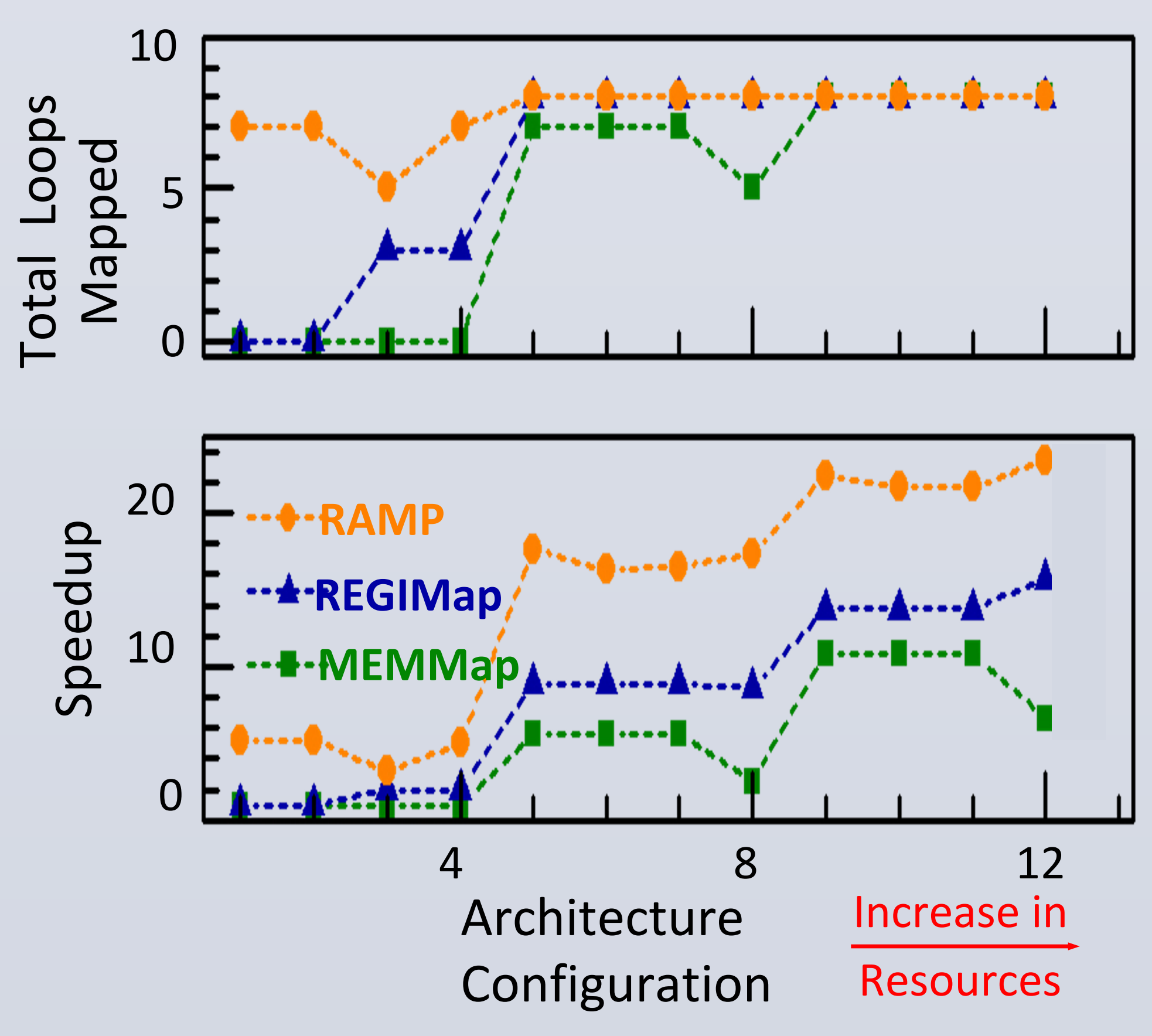
- Partition Mapping Problem in ***3 Sub-Problems***
- **Systematically and Flexibly Explore Resources to Achieve Mapping, Adapting to the Application Needs**
  - ➢ E.g. we can choose to first map the DDG with routing via registers. Then, for any unmapped data dependency, explore different routing options, per failure analysis.

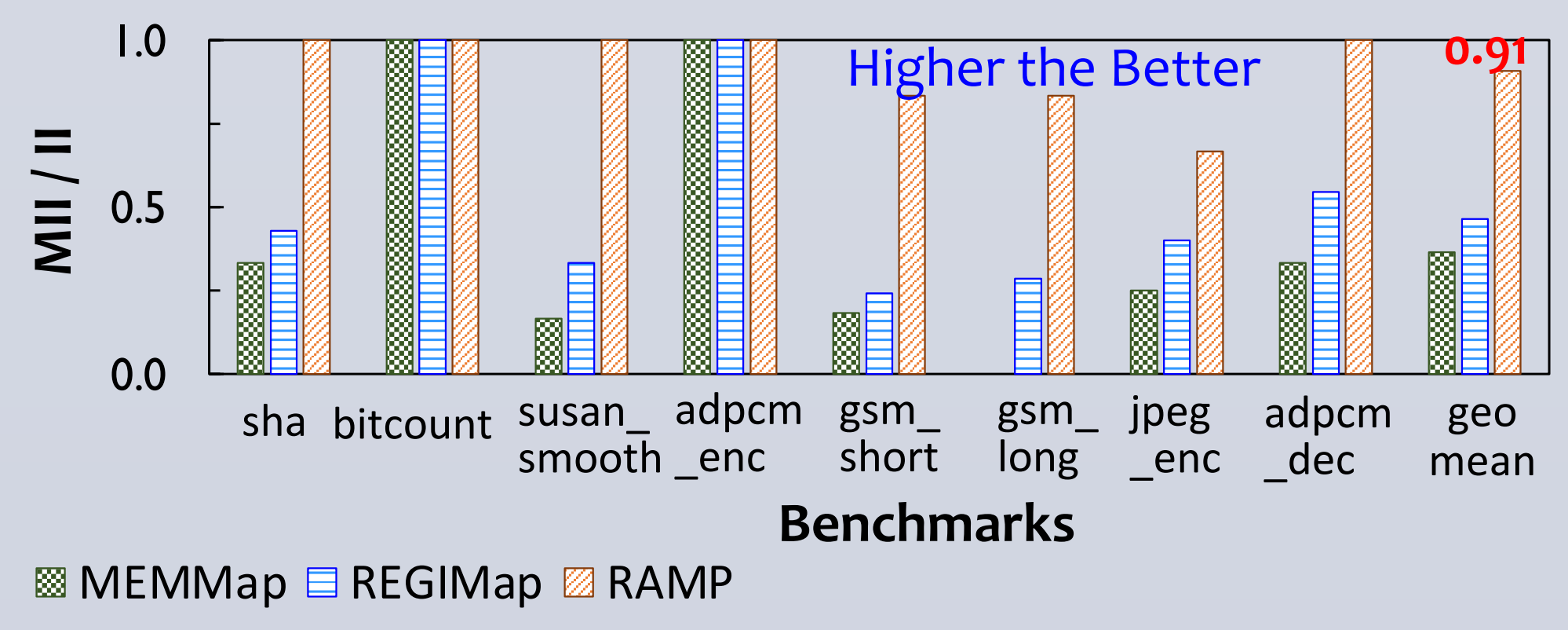

A
Modulo Schedule DDG (Arch., II)
Generate Compatibility Graph CG
P&R (Find Maximal Clique of CG)
B
All Operations Mapped?
Yes
H
Done
Modified & rescheduled DDG
C
No
For Each Unmapped Node v/Edge e
Exhaustive Resource Exploration
D
Failure Analysis
Dependent Ops Scheduled Far?
E
Is pred(v) live-in?
Ops Scheduled at Consequent Time?
F
Graph Modification
Re-schedule
Place & Route
Spill to Mem
Route via PEs and/or Reg.
Spill to distrib. RFs
Load Read-Only Data
Re-Compute
Route via PE
Change Schedule Time
G
Select the Strategy that Uses Minimum Resources

Spilling to Distributed RFs



RAMP Improves CGRA's Acceleration Capability by 2.13x





Experimental Setup

- 8 MiBench benchmarks (top performance-critical loops)
- RAMP modeled using **CCF Compilation & Simulation Framework Available at: https://github.com/cmlasu/ccf** (LLVM 4.0 and gem5 as foundation)
  - ➢ CGRA modeled as a separate core coupled with ARM Cortex-like core
- Evaluation over **12 architectural configurations**
  - ➢ PEs connected in a 2D torus, perform fixed-point computations
  - ➢ CGRA accesses 4 kB data memory and 4 kB instruction memory
  - ➢ Configurations vary in terms of array size, PE functionality, registers etc.

Acknowledgements

This work was partially supported by funding from NSF grants CNS 1525855 and CCF 172346 - NSF/Intel joint research center for Computer Assisted Programming for Heterogeneous Architectures (CAPA).