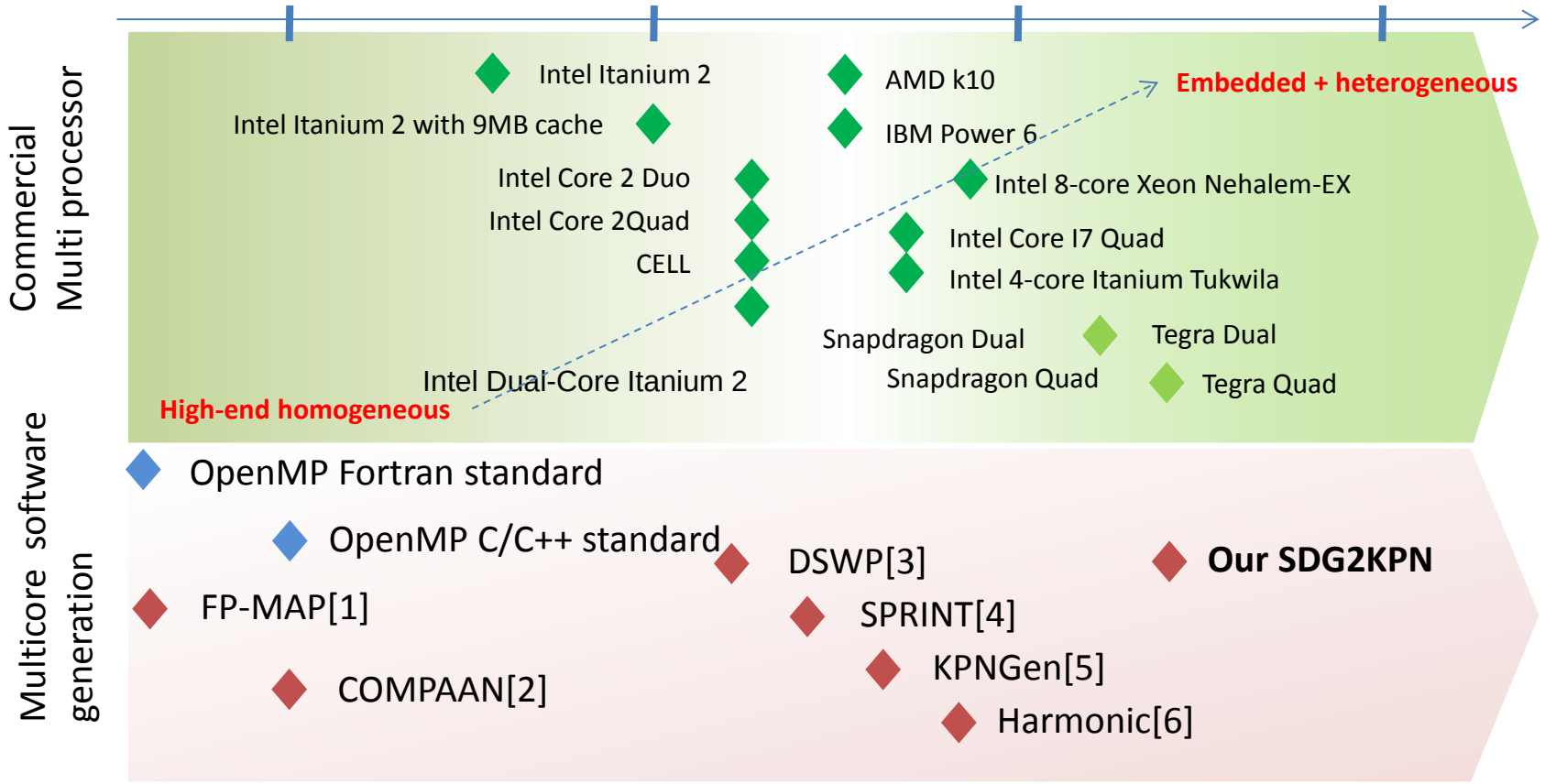
SDG2KPN: System Dependency Graph to Function-level KPN generation of Legacy Code for MPSoCs

Jude Angelo Ambrose, Jorgen Peddersen, Alvin Labios,
Yusuke Yachide, Sri Parameswaran

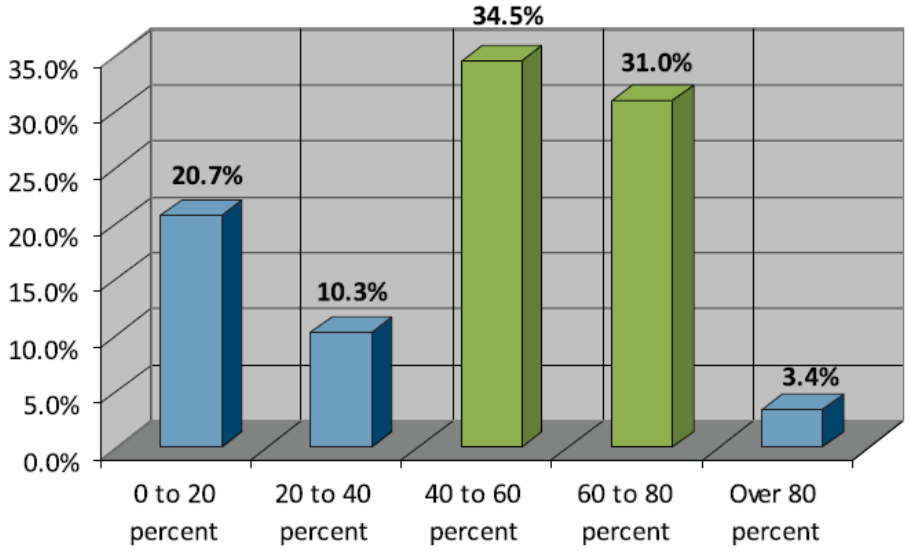
MPSoCs


Commercial
Multi processor
Intel Itanium 2
Intel Itanium 2 with 9MB cache
Intel Core 2 Duo
Intel Core 2Quad
CELL
Intel Dual-Core Itanium 2
AMD k10
IBM Power 6
Intel 8-core Xeon Nehalem-EX
Intel Core I7 Quad
Intel 4-core Itanium Tukwila
Snapdragon Dual
Snapdragon Quad
Tegra Dual
Tegra Quad
Embedded + heterogeneous
High-end homogeneous
Multicore software generation
OpenMP Fortran standard
OpenMP C/C++ standard
FP-MAP[1]
COMPAAN[2]
DSWP[3]
SPRINT[4]
KPNGen[5]
Harmonic[6]
Our SDG2KPN

Legacy Code Usage

- Legacy codes/systems
  - Around 70% of the people claim that they have more than 50% legacy systems

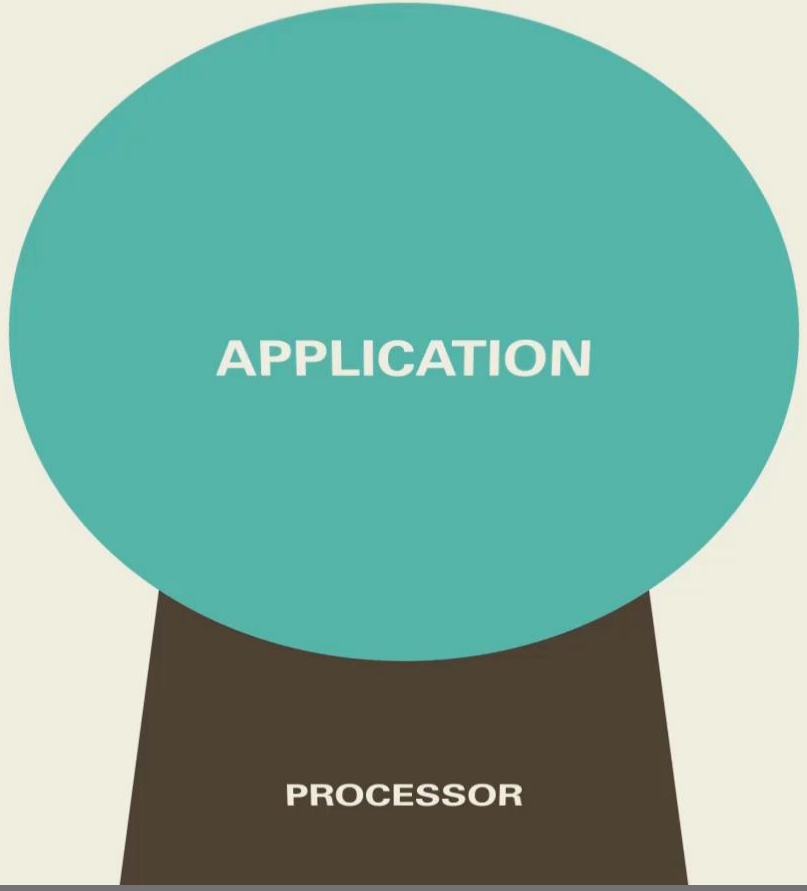
Figure 1. Percentage of IT systems states label as "Legacy Systems; N=29



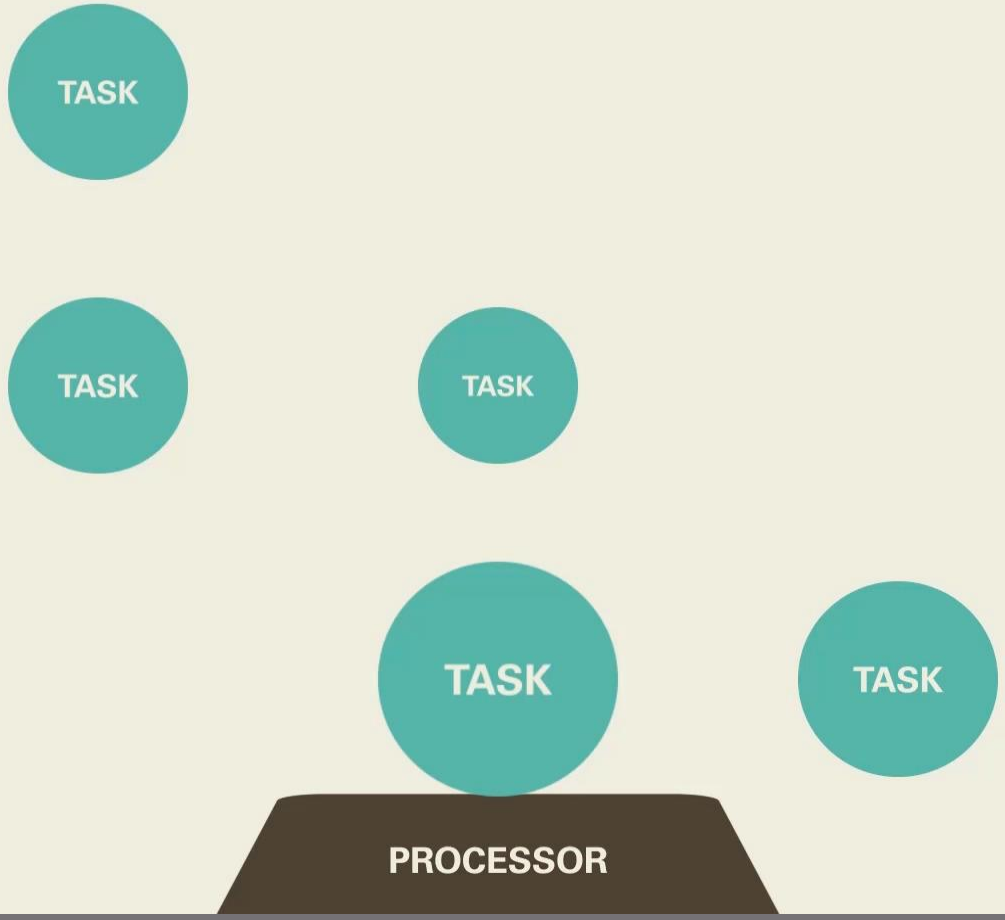
Source: NASCIO's 2008 National Survey on Legacy Systems and Modernization in the States

- However, porting the existing code for other platforms with effective performance is much difficult because of unfamiliar code
  - Challenge is to parallelise the legacy code which was not designed for parallelism

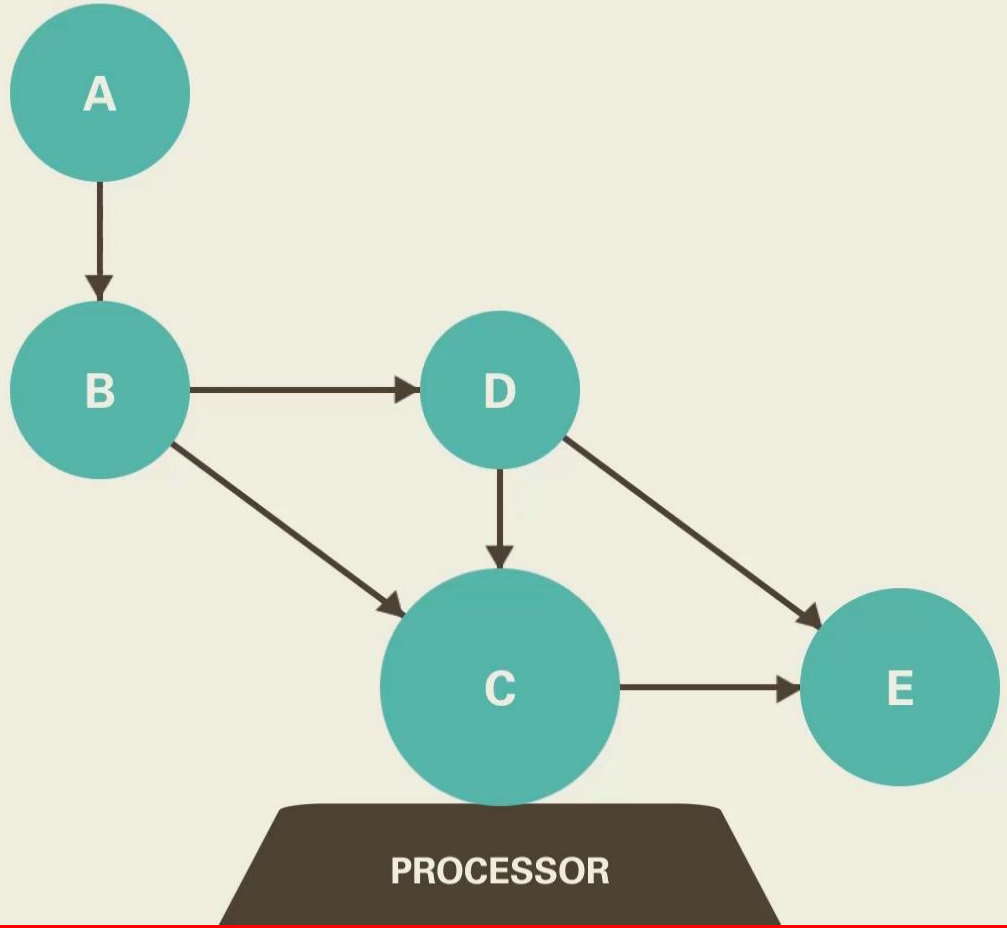
Traditional Processing



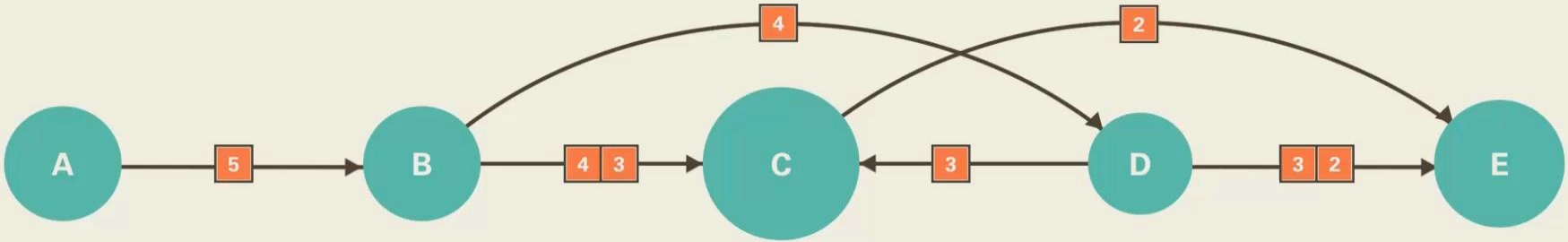
Traditional Processing



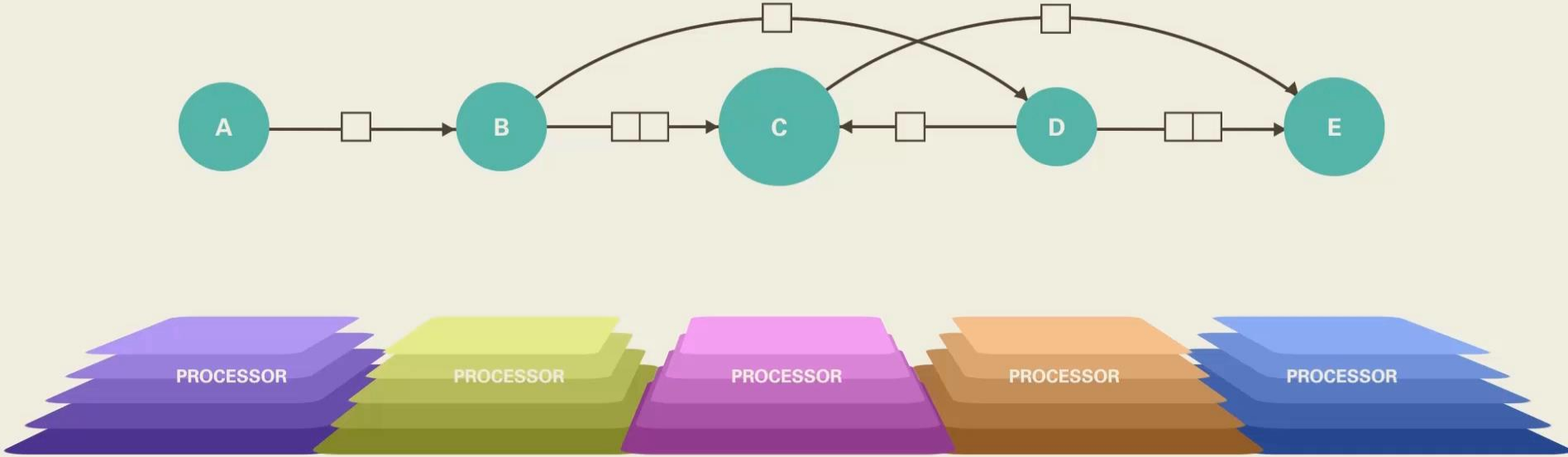
Software Balancing using Pipeline



Choosing Hardware Configurations



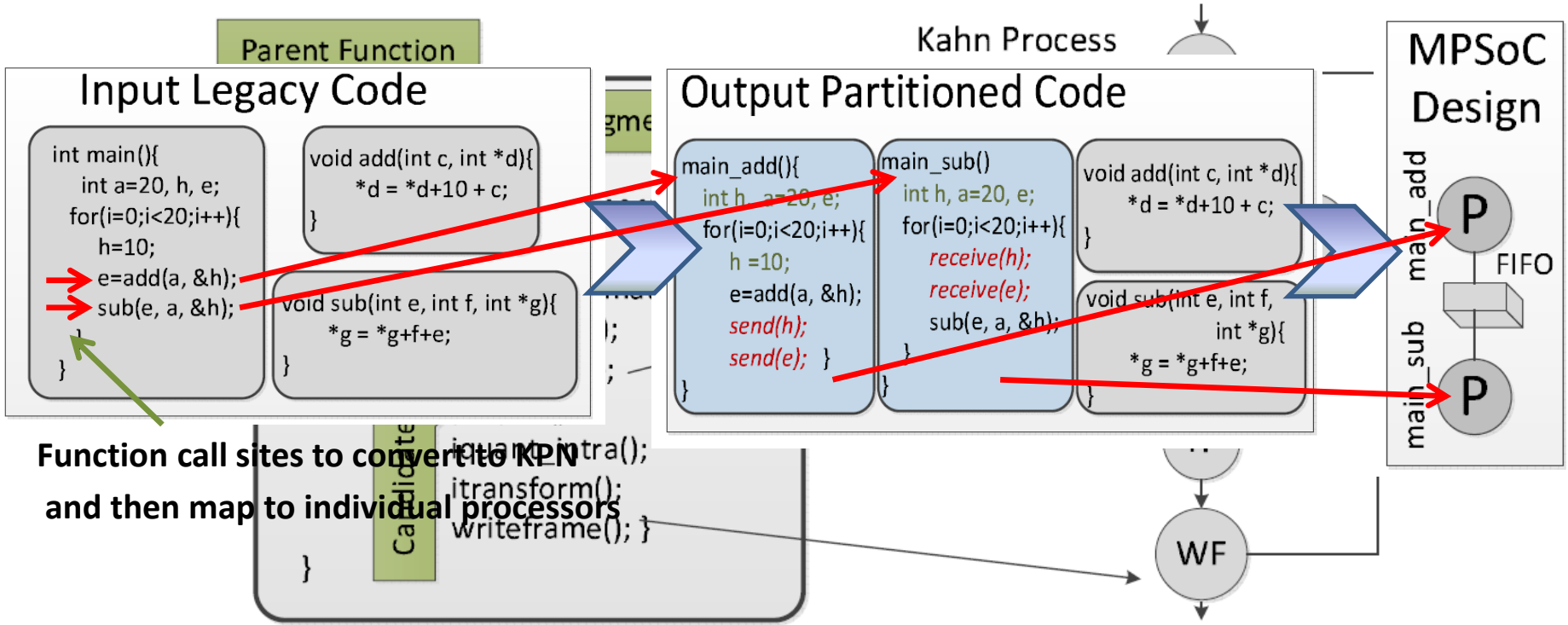
Hardware Balancing for Pipeline



Overview

- Our goal
- Related work
- SDG2KPN Methodology
- Experiments
- Conclusion

Our Goal: Legacy Code to Function-level KPN to Pipeline MPSoC

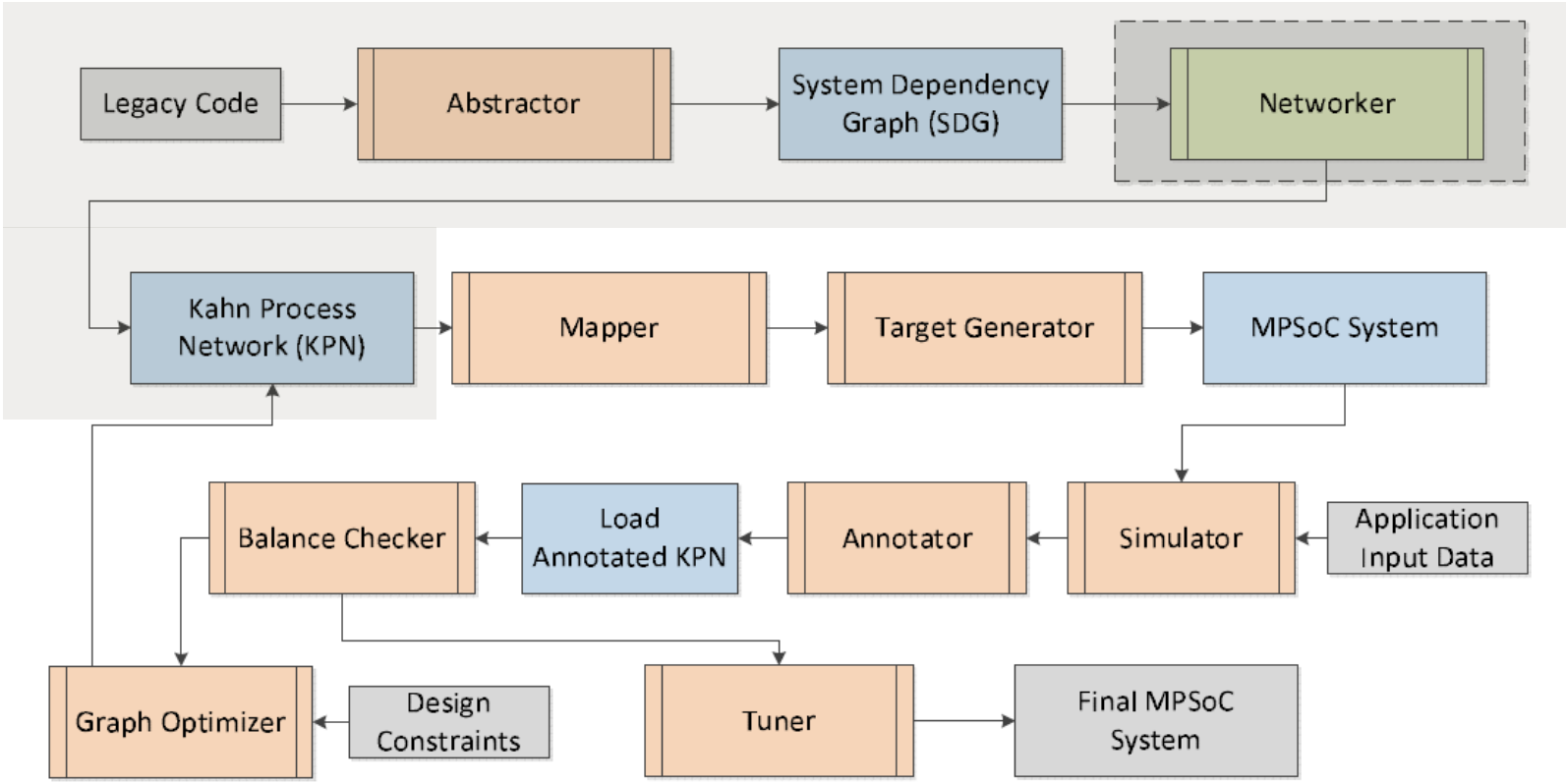


Related Work

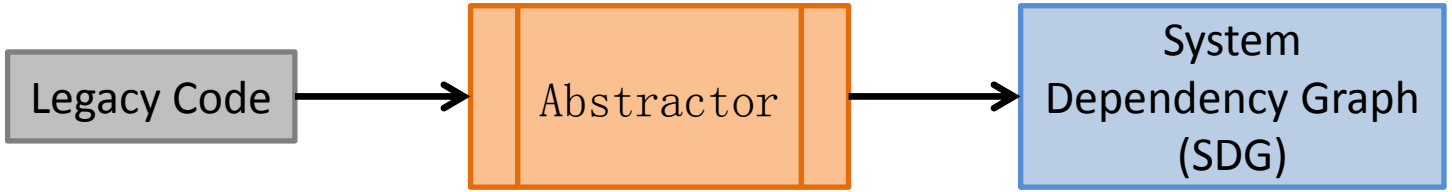
- **KPNGen from the Daedalus framework** (Nikolov et al., DAC'08)
  - ✓ KPN is generated by analysing the source code
  - × manual modification required to the source code
  - × code and MPSoC platform not automatically generated
  - × shared variables not supported
- **COMPAAN** (Kienhuis et al., CODES'00)
  - ✓ a commercial tool, generating multicore code
  - × supporting only Affine Nested Loop Programs using Polyhedral technique
  - × shared variables, such as globals and pointers are not supported

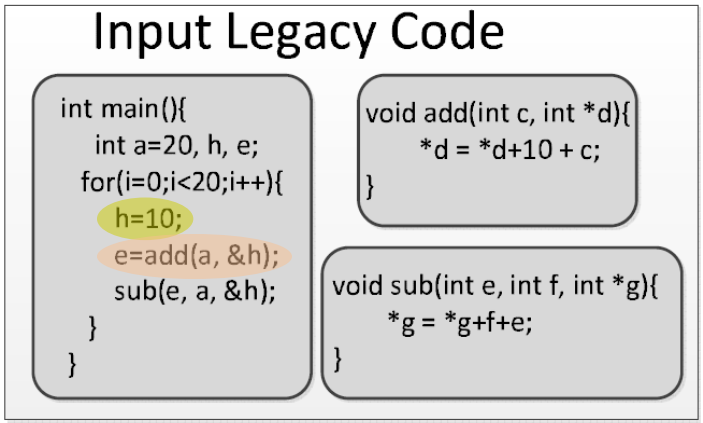
Our SDG2KPN methodology utilizes a rule-based traversal (static analysis) of the SDG of <u>any</u> legacy code to find dependencies between functions, supporting analysis of <u>shared variables</u> which was hitherto not supported

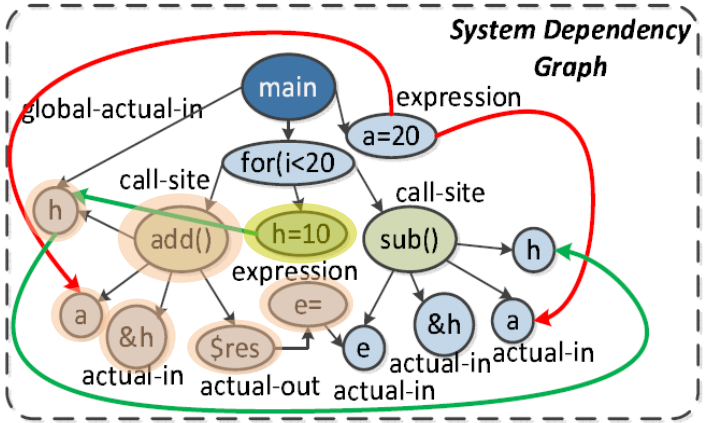
SDG2KPN Flow



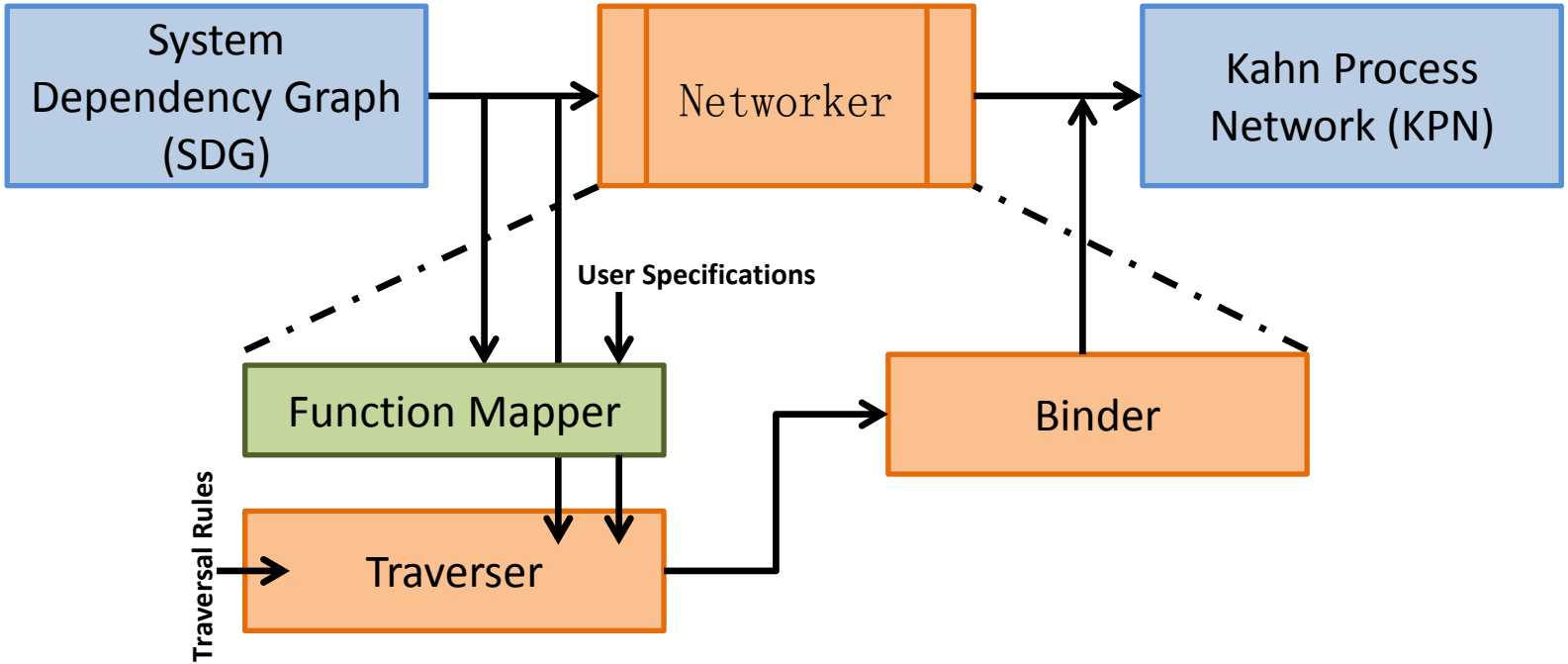
SDG Creation





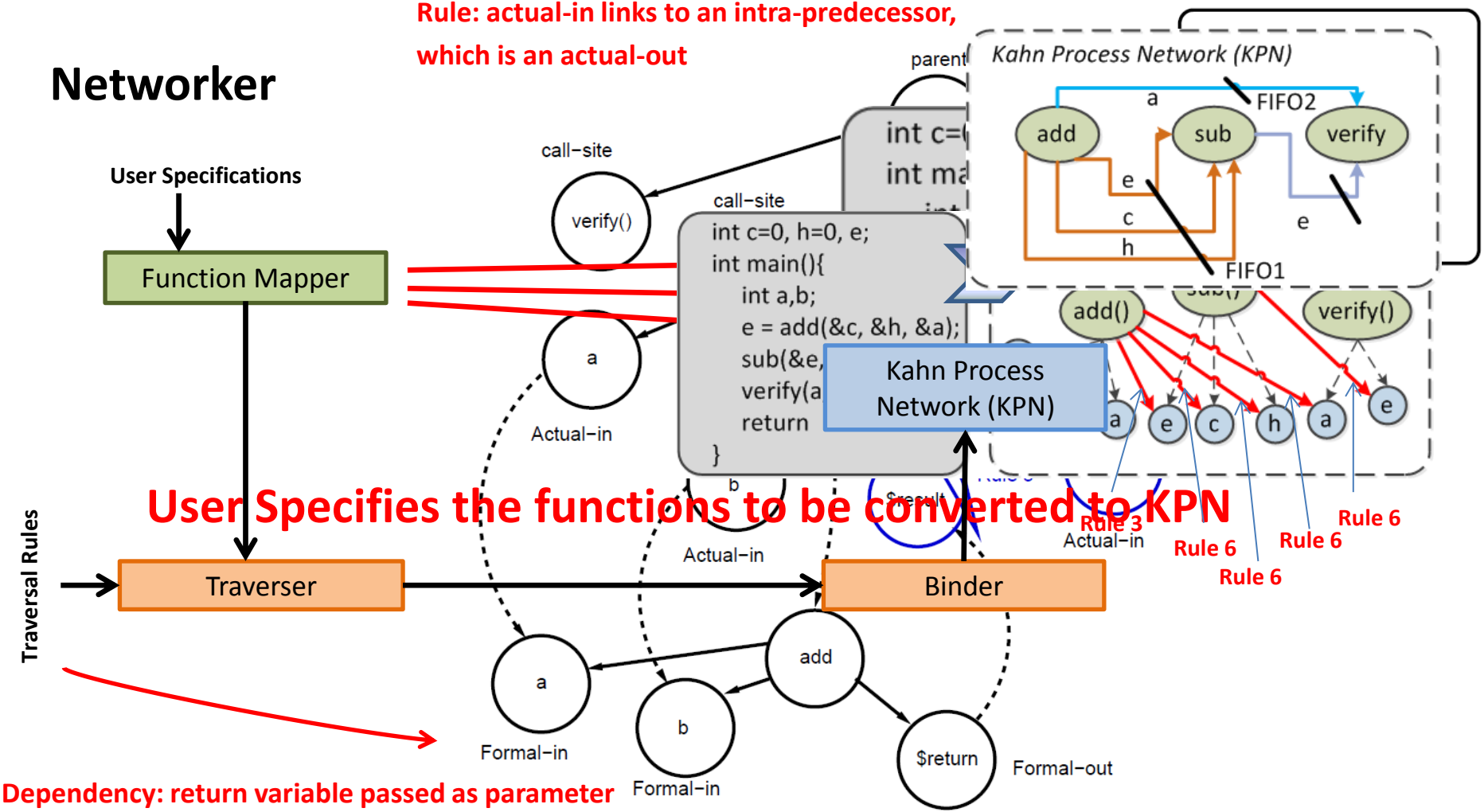


KPN Generation



Networker

Rule: actual-in links to an intra-predecessor, which is an actual-out



User Specifies the functions to be converted to KPN

Dependency: return variable passed as parameter

Experiments – Rule Evaluation

Apps	No. KPN Nodes	No. of Rules	Generation Time (sec)	Estimated Manual Gen.	LOC
AES	2	16	4.8	2 days	950
MPEG-2 (enc)	10	97	21.4	2 months	8k
MJPEG	6	75	8.0	3 weeks	2k
H.264 (enc)	9	1064	164.9	6 months	58k
ADPCM (enc)	6	27	4.3	1 week	285

Experiments – MPSoC Executions

Apps	No. of Processors	Latency KPN (Mcycles)	Latency single (Mcycles)	Power KPN (mW)	Power single (mW)
AES	2	131 ← 3.4%	135	180.54	123.34
MJPEG (f)	6	445 ← 16.8%	536	494.39	113.55
MJPEG (mb)	6	508 ← 5.15%	536	367.02	113.55
ADPCM (enc)	6	224 ← No speedup due to feedbacks	113	579.19	131.25

Conclusion

- a novel SDG to KPN conversion methodology is proposed
- a rule based traversal of the SDG is performed to create dependencies of variables across functions
- all the variable constructs of the legacy code, including shared variables such as globals and pointers, are supported, which was hitherto not possible.

THANK YOU