Technion

RAMONchips

The Plural Architecture

Shared Memory Many-core with Hardware Scheduling

Ran Ginosar

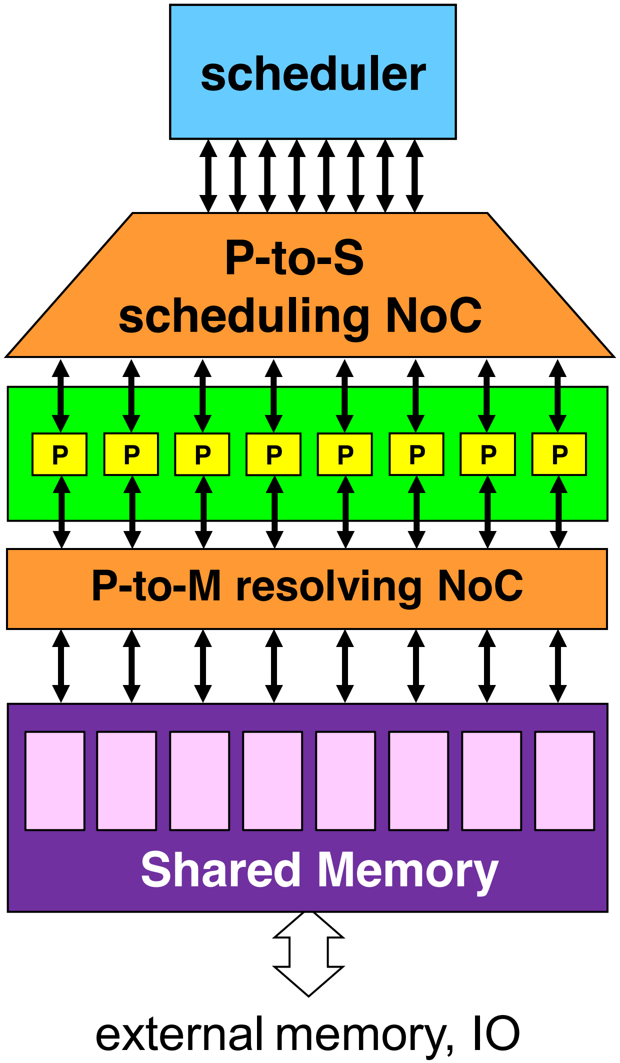
Technion—Israel Institute of Technology

& Ramon Chips, Ltd.

Israel

MPSoC July 2014

The Plural Architecture



Hardware scheduler / dispatcher / synchronizer

Low (zero) latency parallel scheduling
enables fine granularity

Many small processor cores
Small private memories (stack, L1)

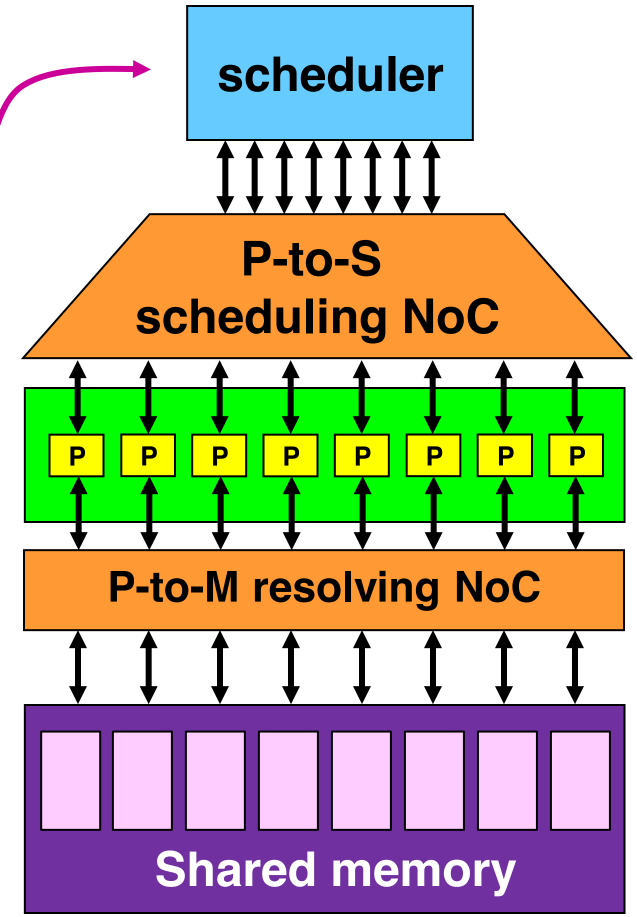
Fast NOC to memory
(Multistage Interconnection Network)
NOC resolves conflicts

SHARED memory, many banks
~Equi-distant from cores (2-3 cycles)

"Anti-local" address interleaving
Negligible conflicts

The Plural task-oriented programming model

- Programmer generates TWO parts:
  - Task-dependency-graph = 'task map'
  - Sequential task codes
- Task maps loaded into scheduler
- Tasks loaded into memory



Task template:

```
⎧ singular    ⎫
⎨ duplicable  ⎬  task xxx( dependencies )
⎩ control     ⎭
{
        … # ….   // # is instance number
        …..
}
```

Fine Grain Parallelization

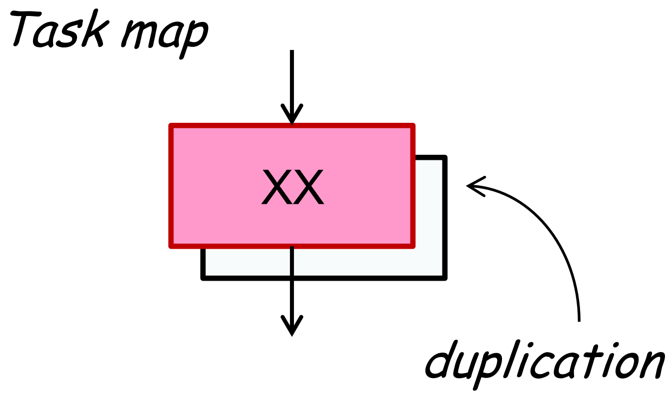
Convert (independent) loop iterations

```
for ( i=0; i<10000; i++ ) { a[i] = b[i]*c[i]; }
```

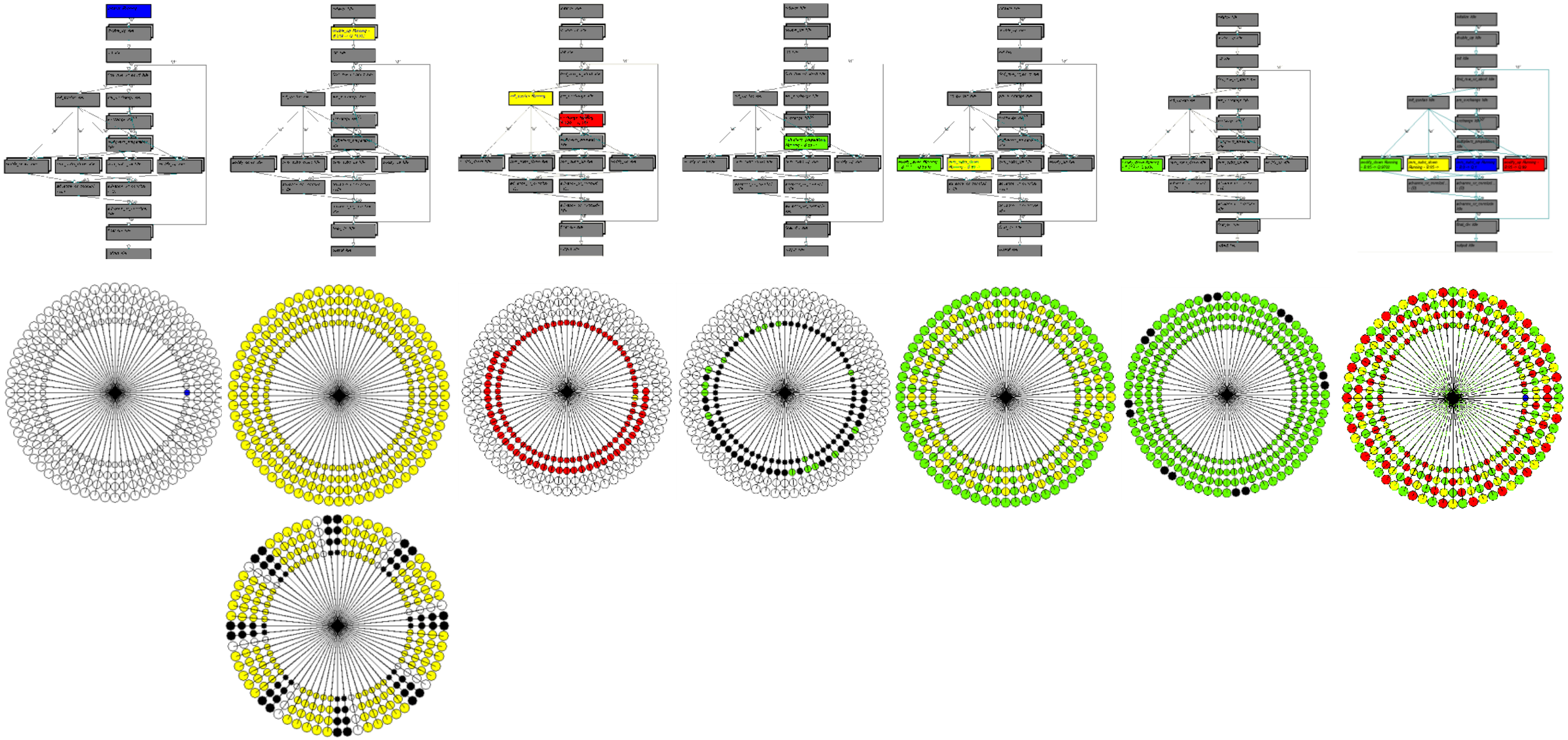
into parallel tasks

```
set_quota (XX,10000)

duplicable task XX
{ a[#] = b[#]*c[#]; }
 // # is instance number
```

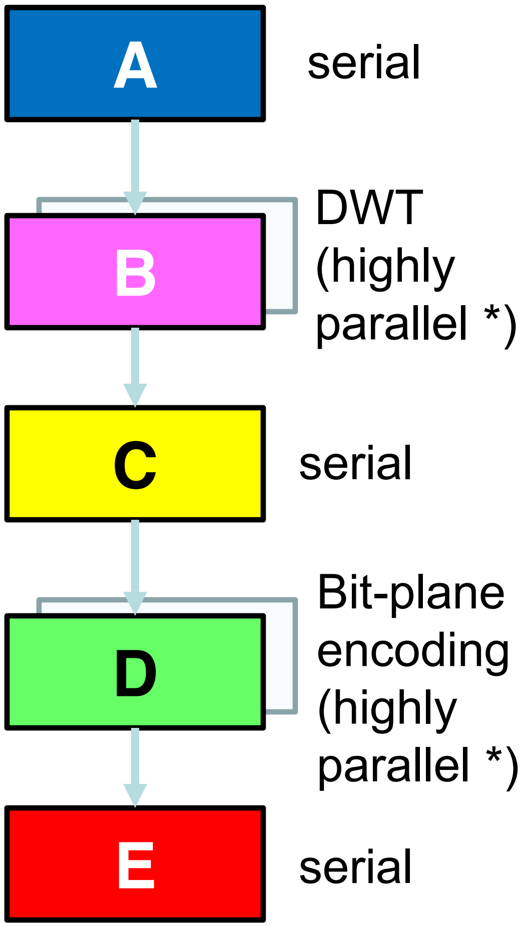


Example: Linear Solver



PIPELINED stream processing: ManyFlow

Example: JPEG2000



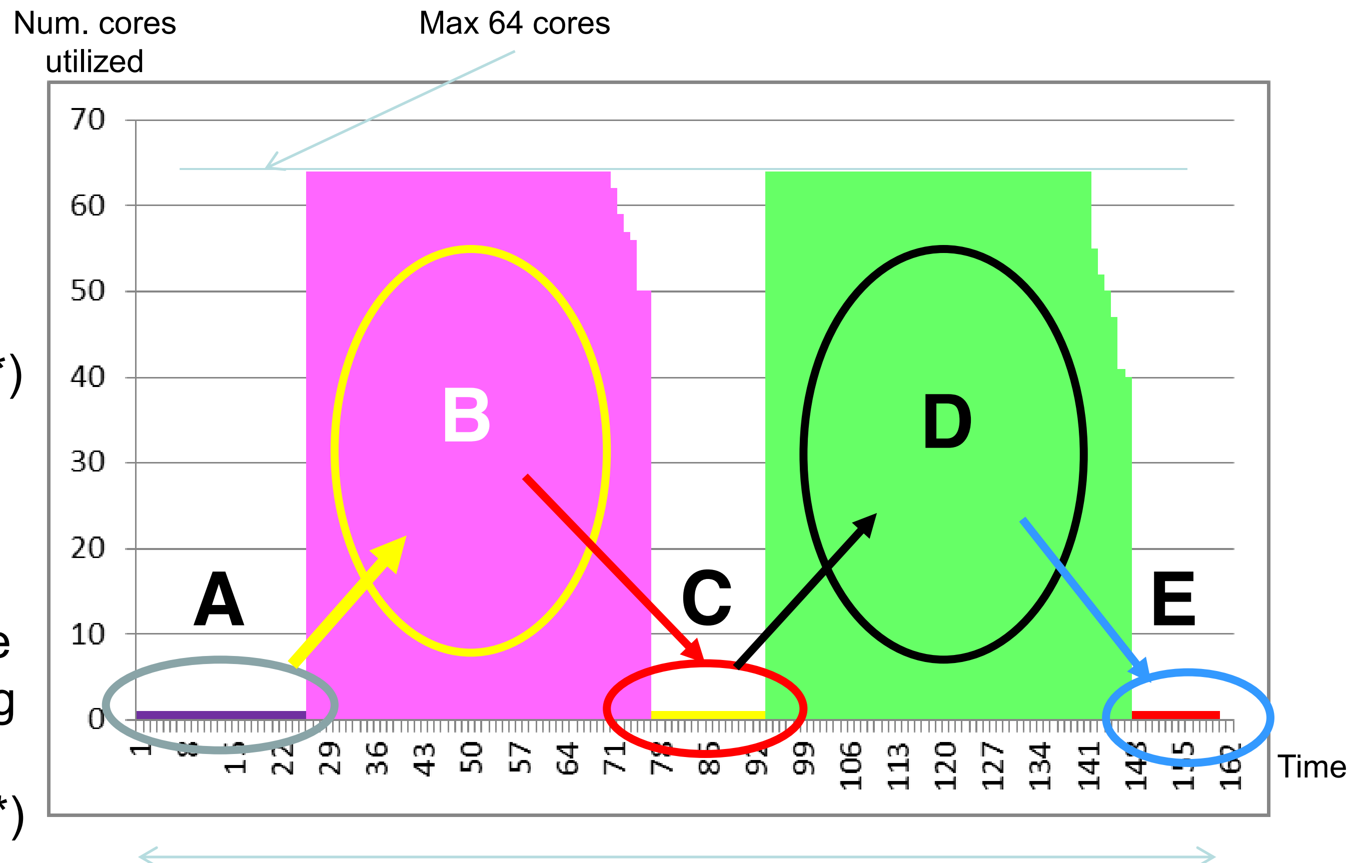
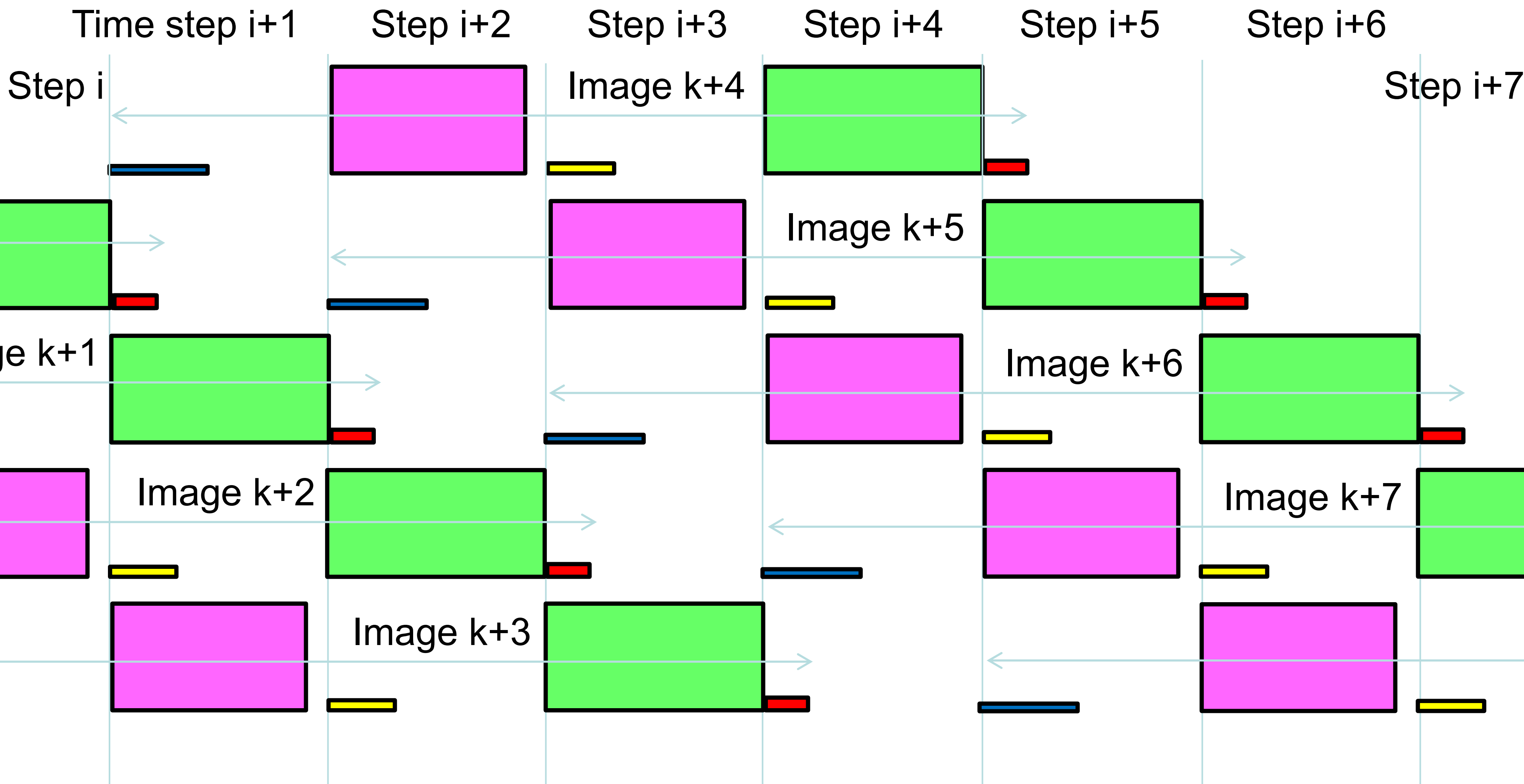


Image compression time: 160 (relative time units)

Low utilization: only 65%

Hardware-like Pipeline

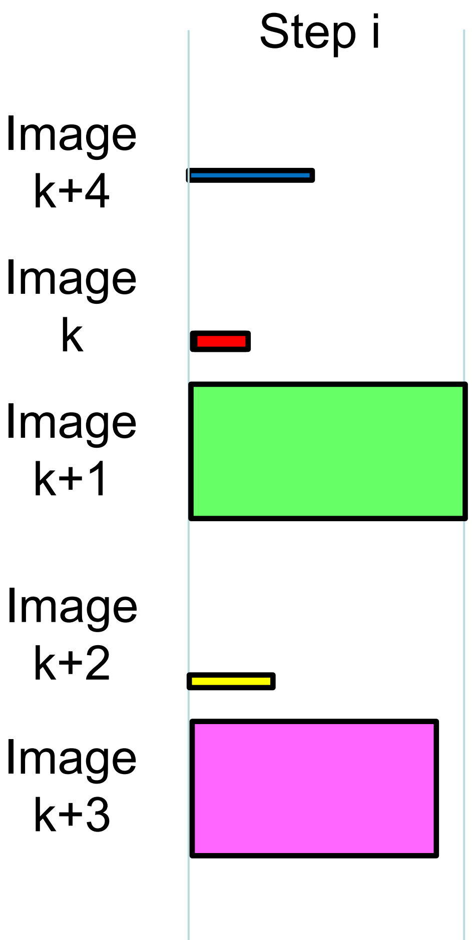


Needs 5 stages: two with 64 cores each, three with one core each (total 131 cores)

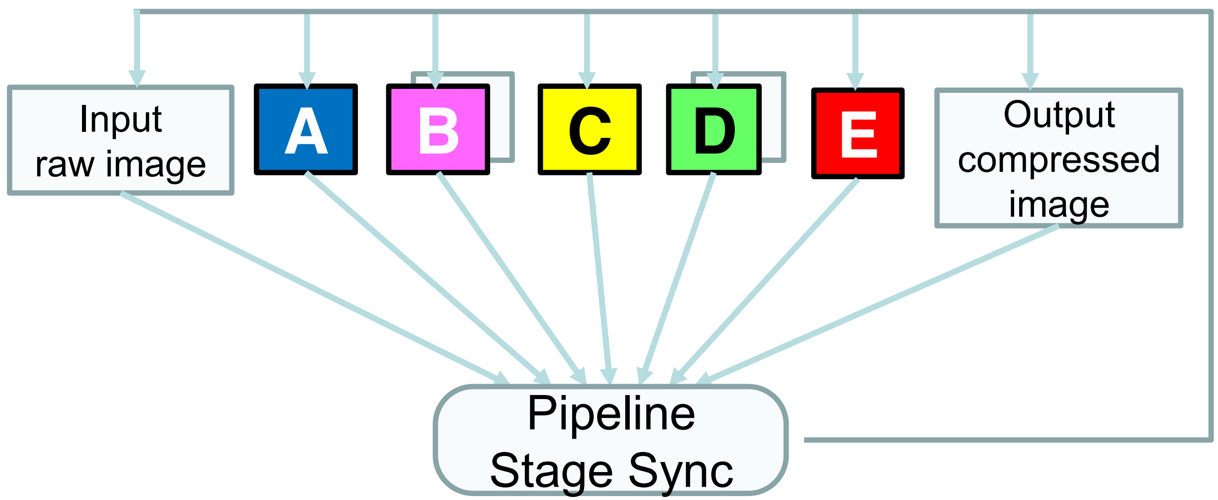
If only 64 cores, time / step = 64x2 + 25 = 153 (how ? What is the utilization?)

Hard to program, inefficient, inflexible, fixed task per core. <u>Need to store 5 images</u>

Parallel / pipelined ManyFlow

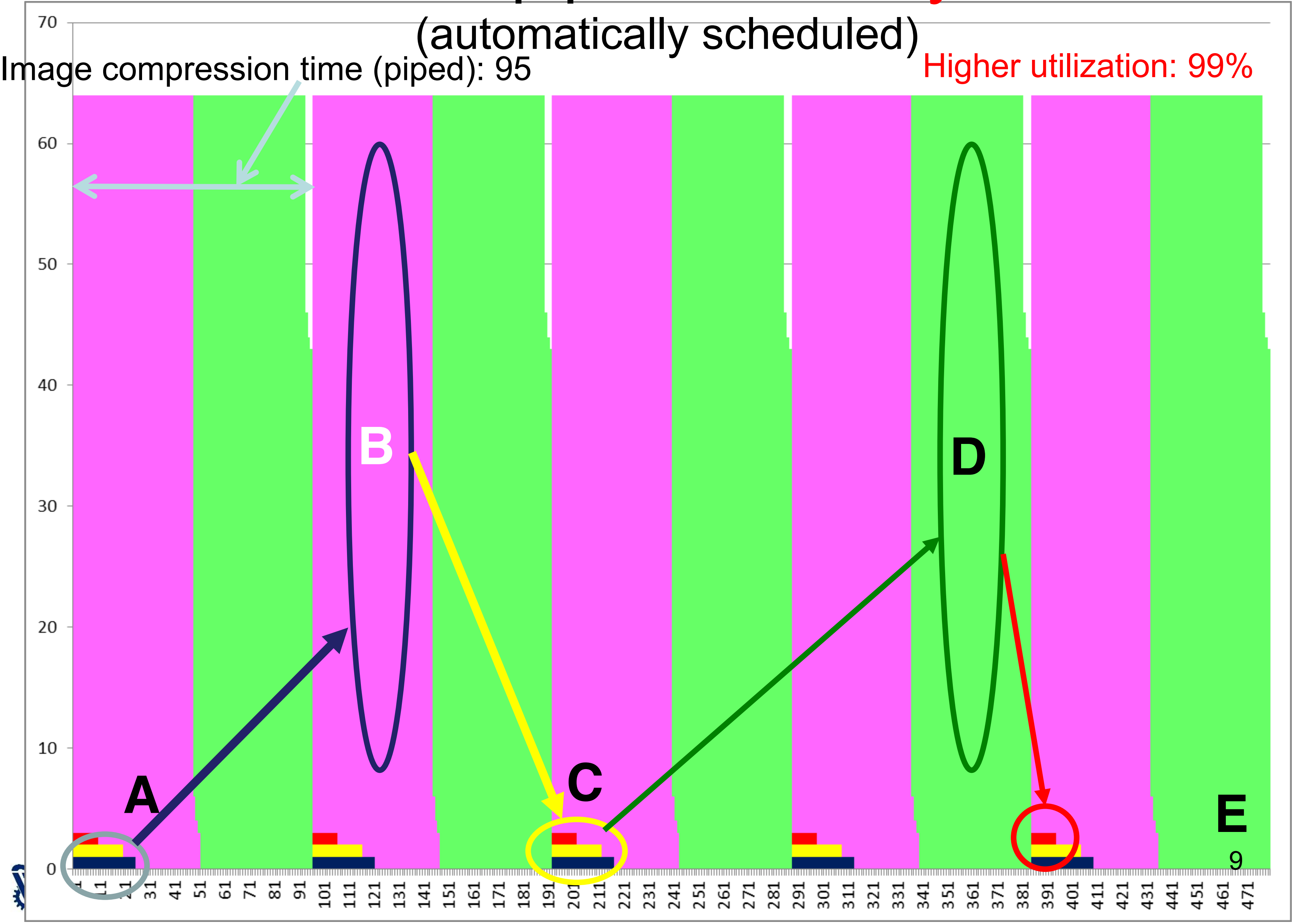


All stages independent (order does not matter)
→ Can run concurrently
→ Scheduler will dispatch most efficiently



Bottleneck: need to store 7 images

Parallel / pipelined ManyFlow



The Plural Architecture: Some benefits

- Shared, uniform (~equi-distant) memory
  - no worry which core does what
  - no advantage to any core because it already holds the data
- Many-bank memory + fast P-to-M NoC
  - low latency
  - no bottleneck accessing shared memory
- Fast scheduling of tasks to free cores (many at once)
  - enables fine grain data parallelism
  - harder in other architectures due to:
    - task scheduling overhead
    - data locality
- Any core can do any task equally well on short notice
  - scales well
- Programming model:
  - PRAM-like
  - intuitive to programmers
  - “easy” for automatic parallelizing compiler & formal verification (?)

Summary

- Simple many-core architecture
  - Inspired by PRAM
- Hardware scheduling
- Task-based programming model
- Designed to achieve the goal of ‘more cores, less power’