Applied Micro Circuits Corporation

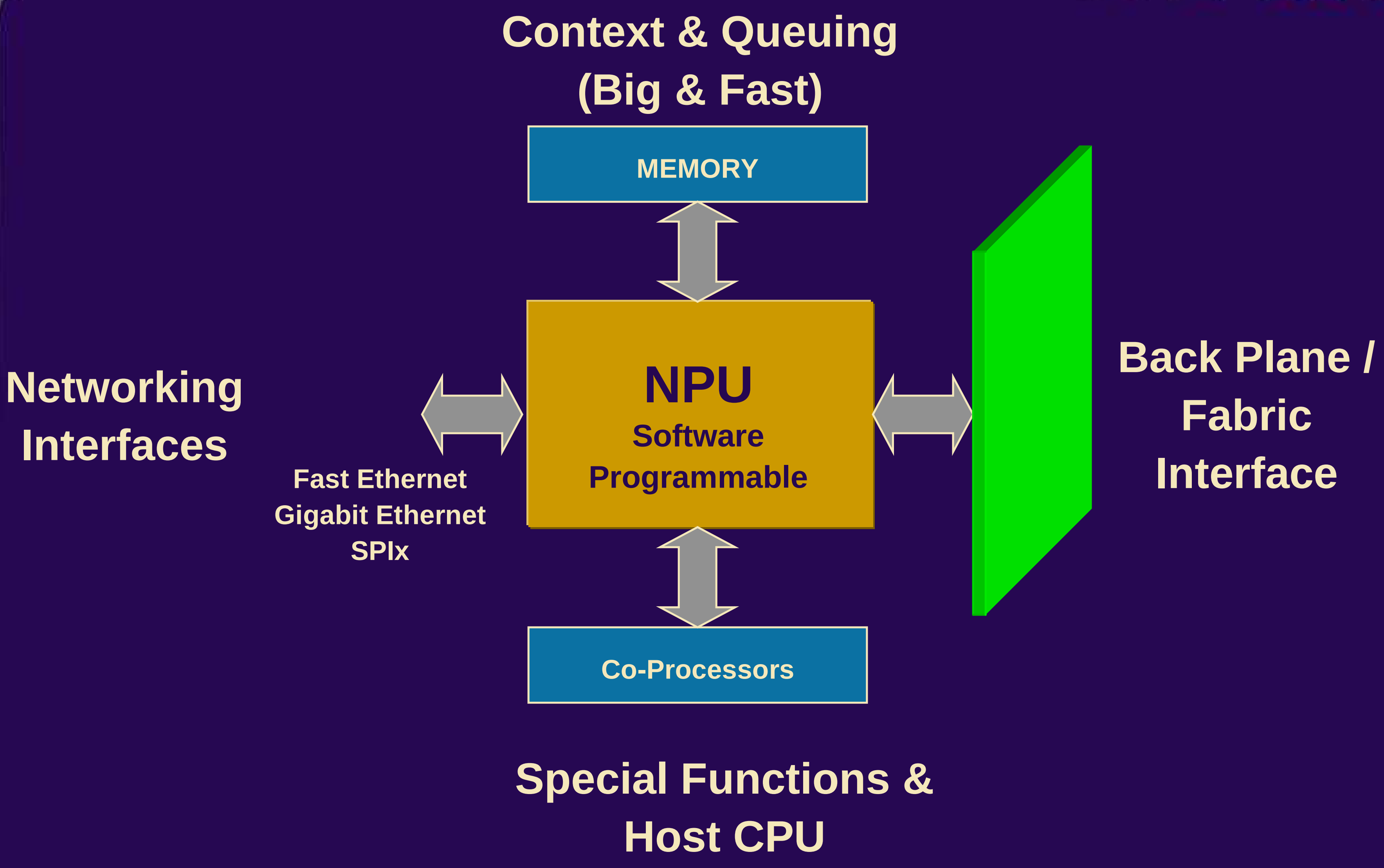
Challenges in Making Highly Integrated Network Processors

Keith Morris
Director of Marketing
Switching & Network Processing
kmorris@amcc.com

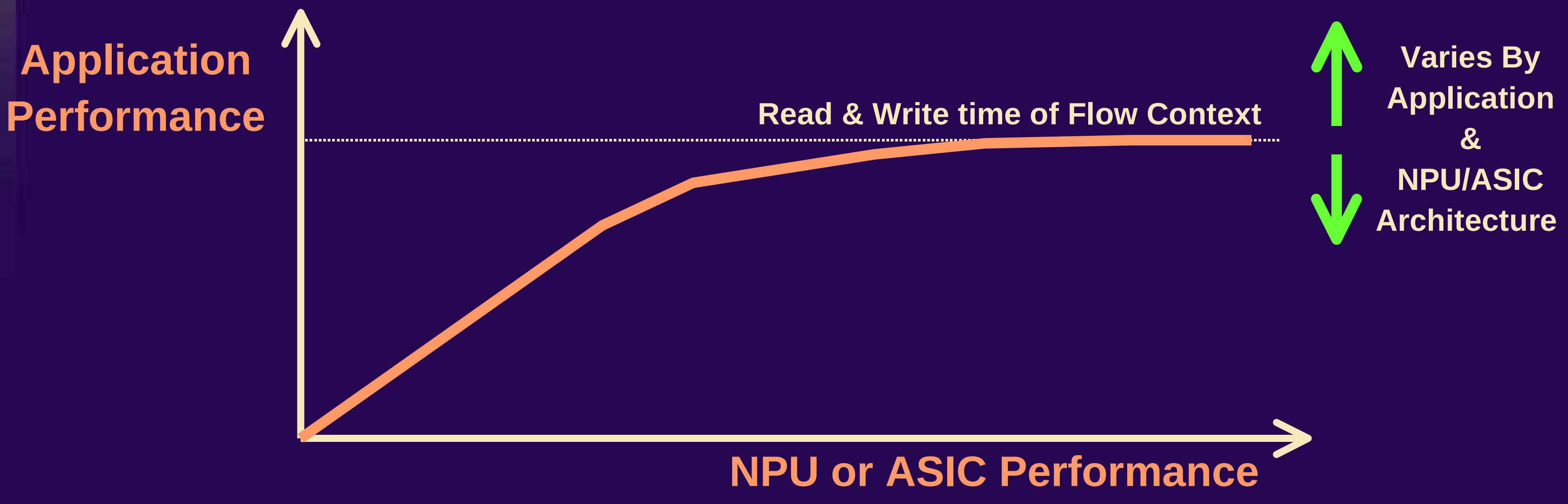
Agenda

- NPU Definition
- Challenges & Solutions
  - Physical
  - Performance
  - Flexibility
  - Scalability & Re-use
- Putting it all together – Integrated NPU

NPU Definition

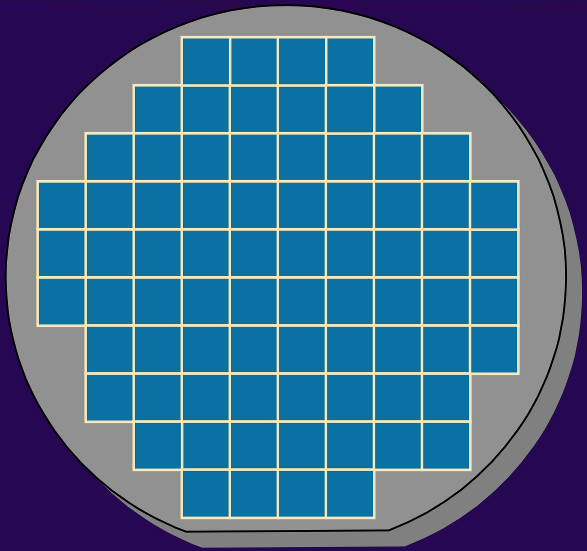


Context Handling – The Ultimate Bottleneck

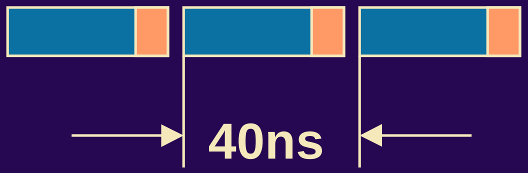


Challenges & Solutions

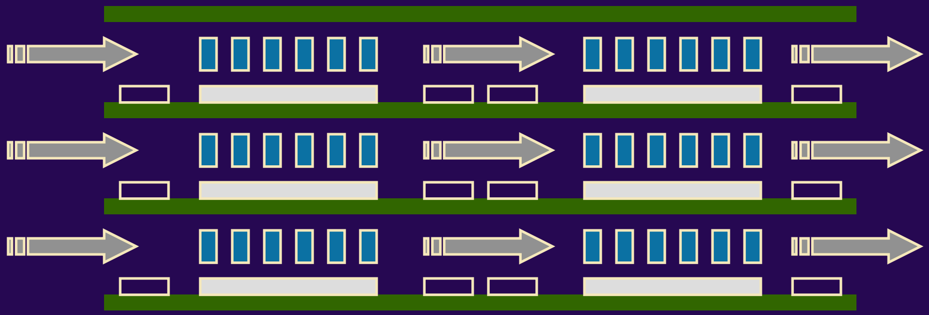
Challenge: Physical Constraints



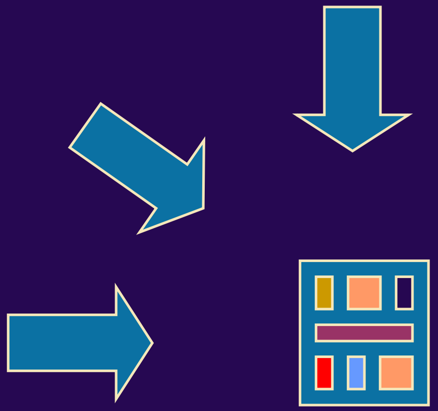
Small Silicon Die Area



Packet Arrival Rate



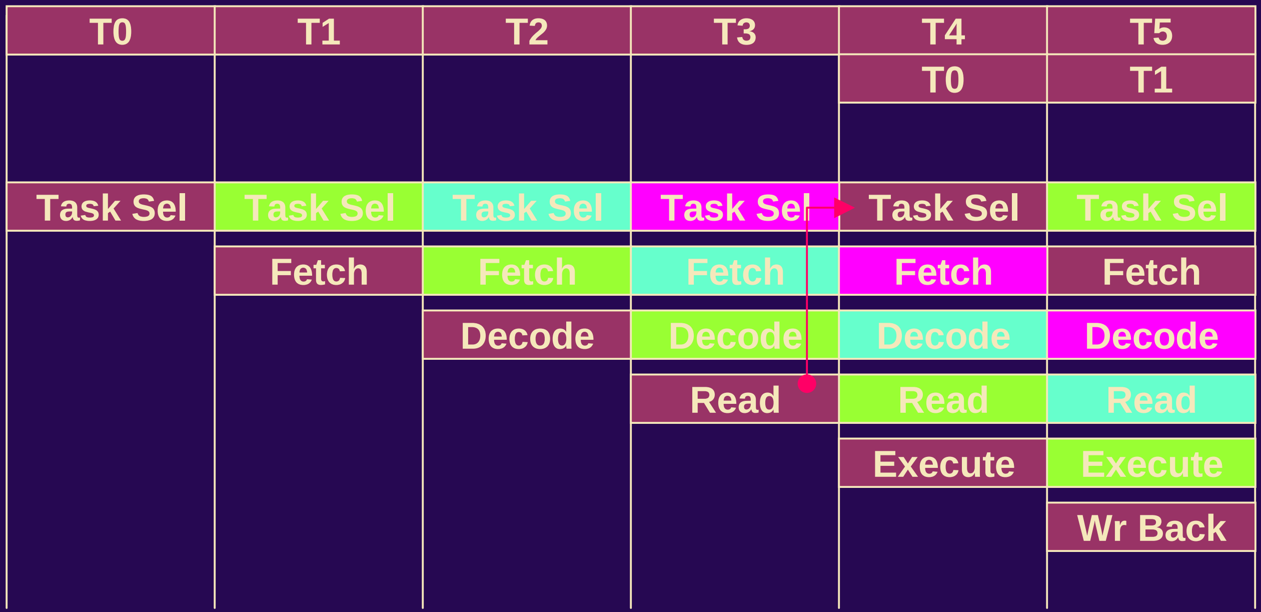
System Power Dissipation



Efficient nPcore Architecture

Networking Optimized Execution Pipeline

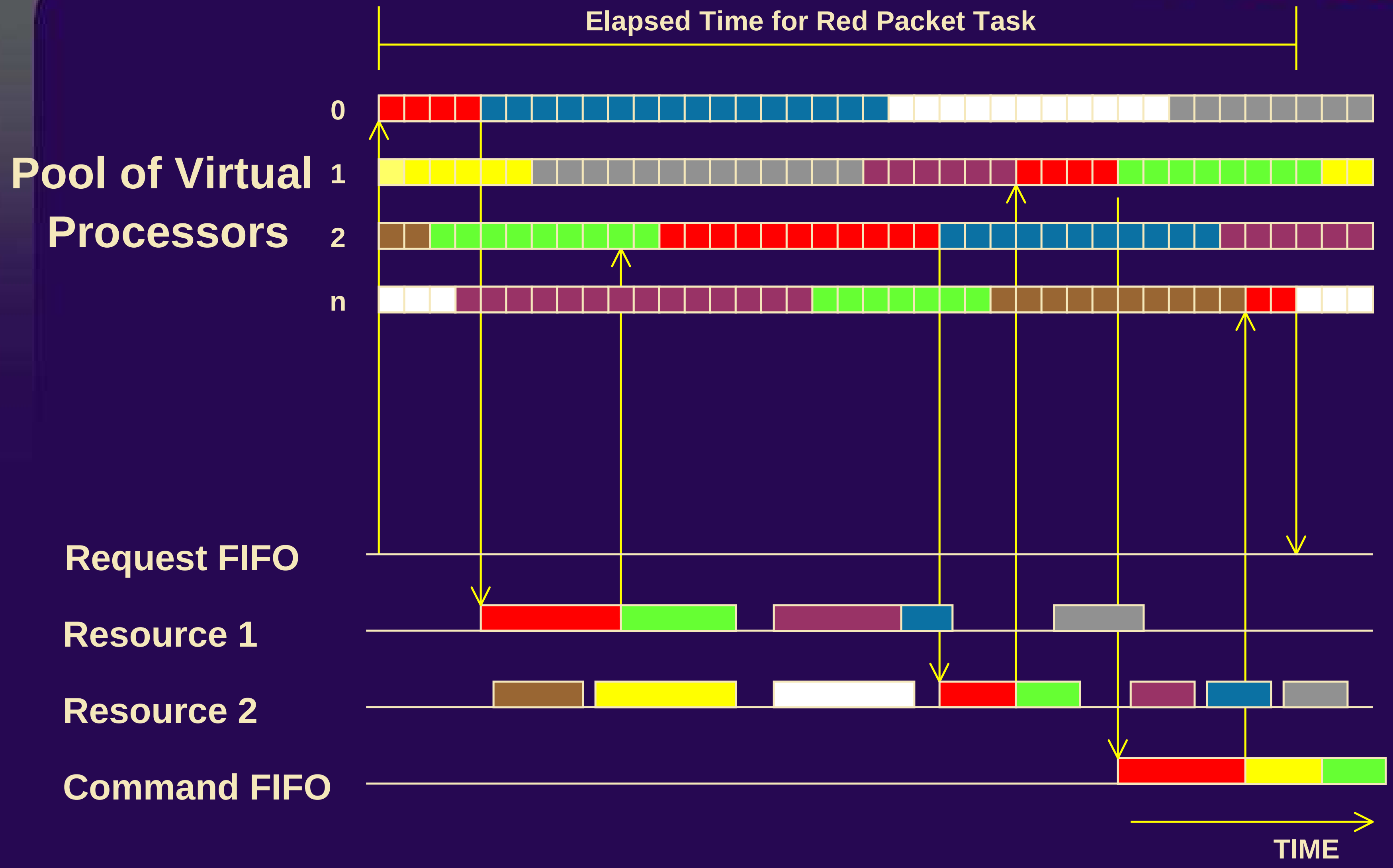
Overlapping Virtual Processors



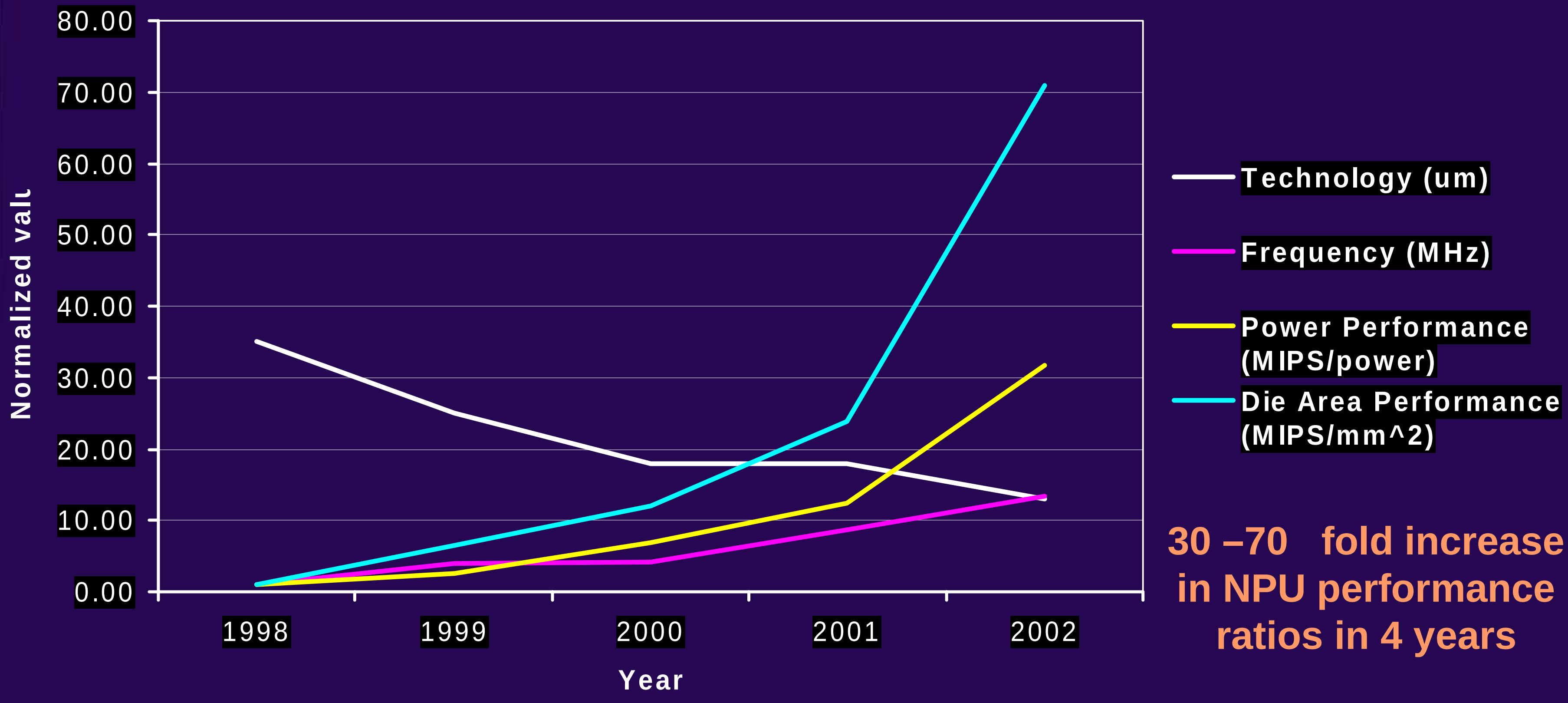
Case Statements and Conditional Jumps do not cause pipeline breakages

Extremely important for deterministic performance in decision rich data path processing

Zero Cycle Task Switching – Hides Latency

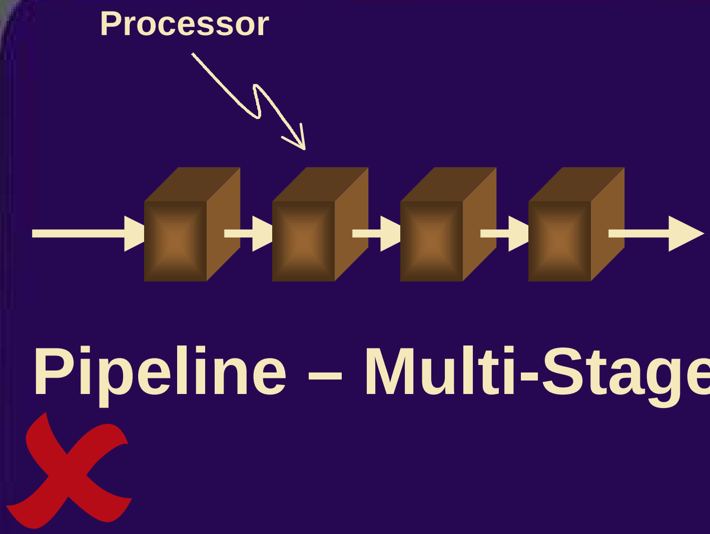


Frequency, Power and Die Area for nPcores



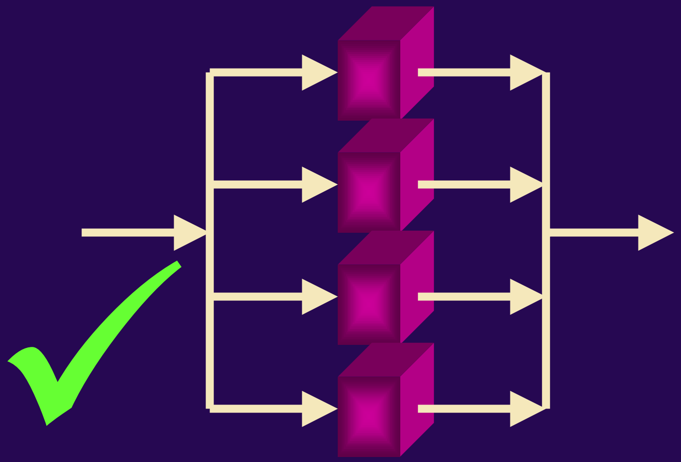
30 –70 fold increase in NPU performance ratios in 4 years

Challenge: Scaling Processing Performance



Pipeline – Multi-Stage

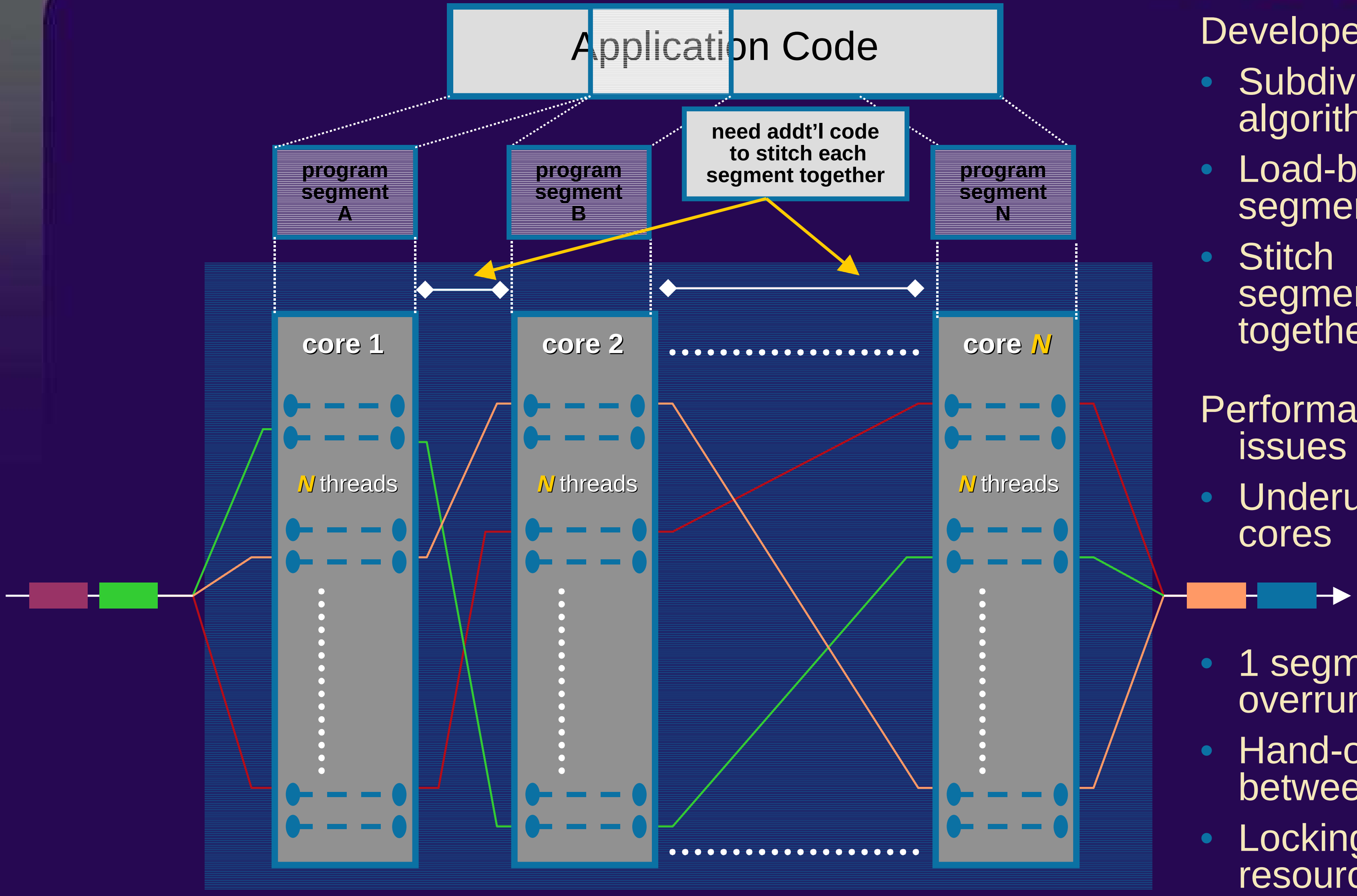
- Complex software partitioning
- Very sensitive to latency
  - Each stage must complete within the smallest packet time
  - One stage becomes the bottleneck
- Poor Scalability
  - Eventually some stages may need to become parallel



Parallel – Single Stage

- Allows simplest programming model
  - Run to completion
- Not sensitive to latency
  - Elapsed time to process a packet can be very long if required
- Highly scalable
  - 100's of tasks, multiple processors

Multi-Stage – Complex and Inefficient



Developer must:

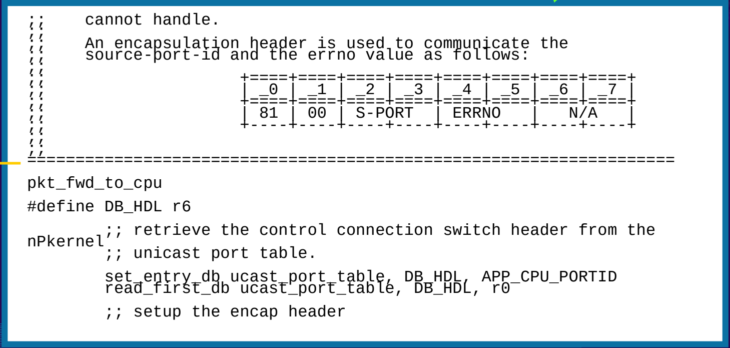
- Subdivide algorithm
- Load-balance segments
- Stitch segments together

Performance issues with:

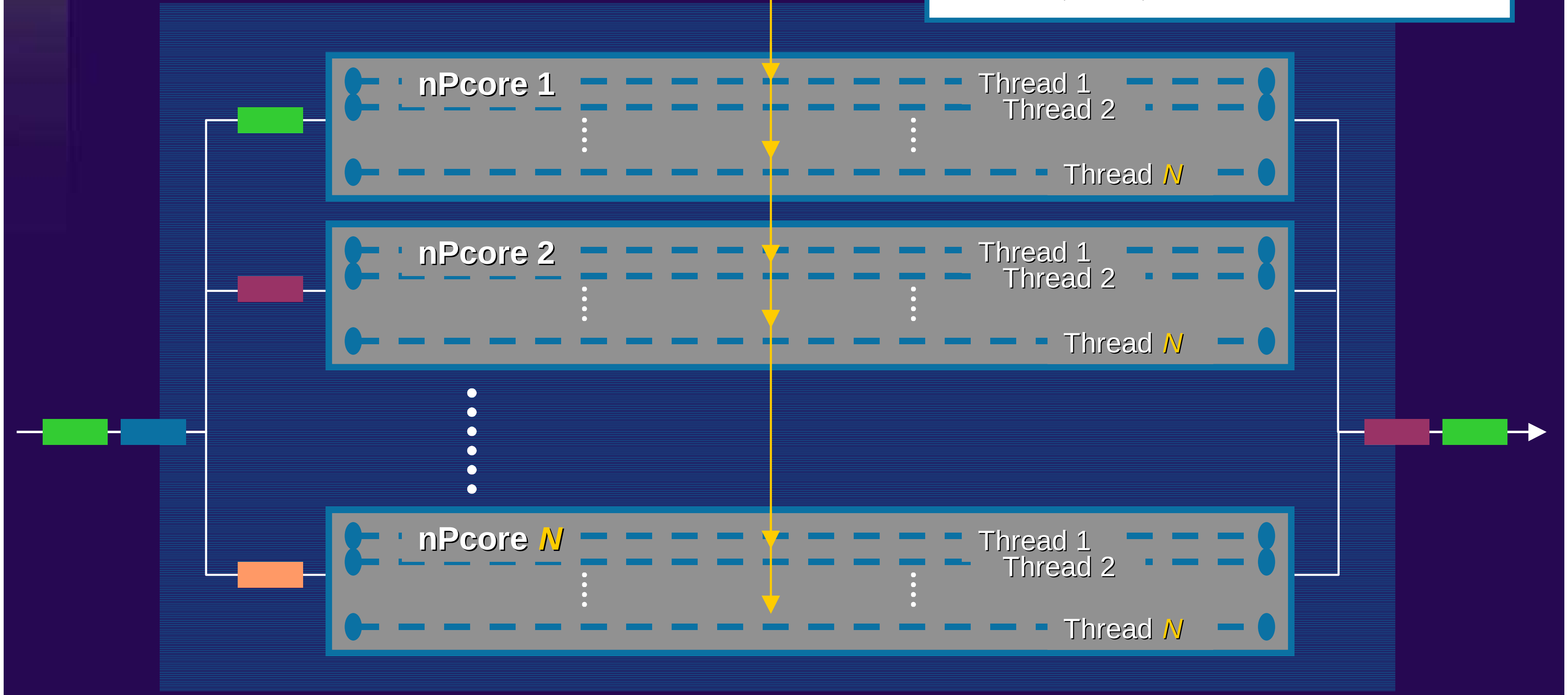
- Underutilized cores
- 1 segment can overrun next
- Hand-offs between cores
- Locking shared resources

AMCC: 1-Stage – Fundamentally Simpler

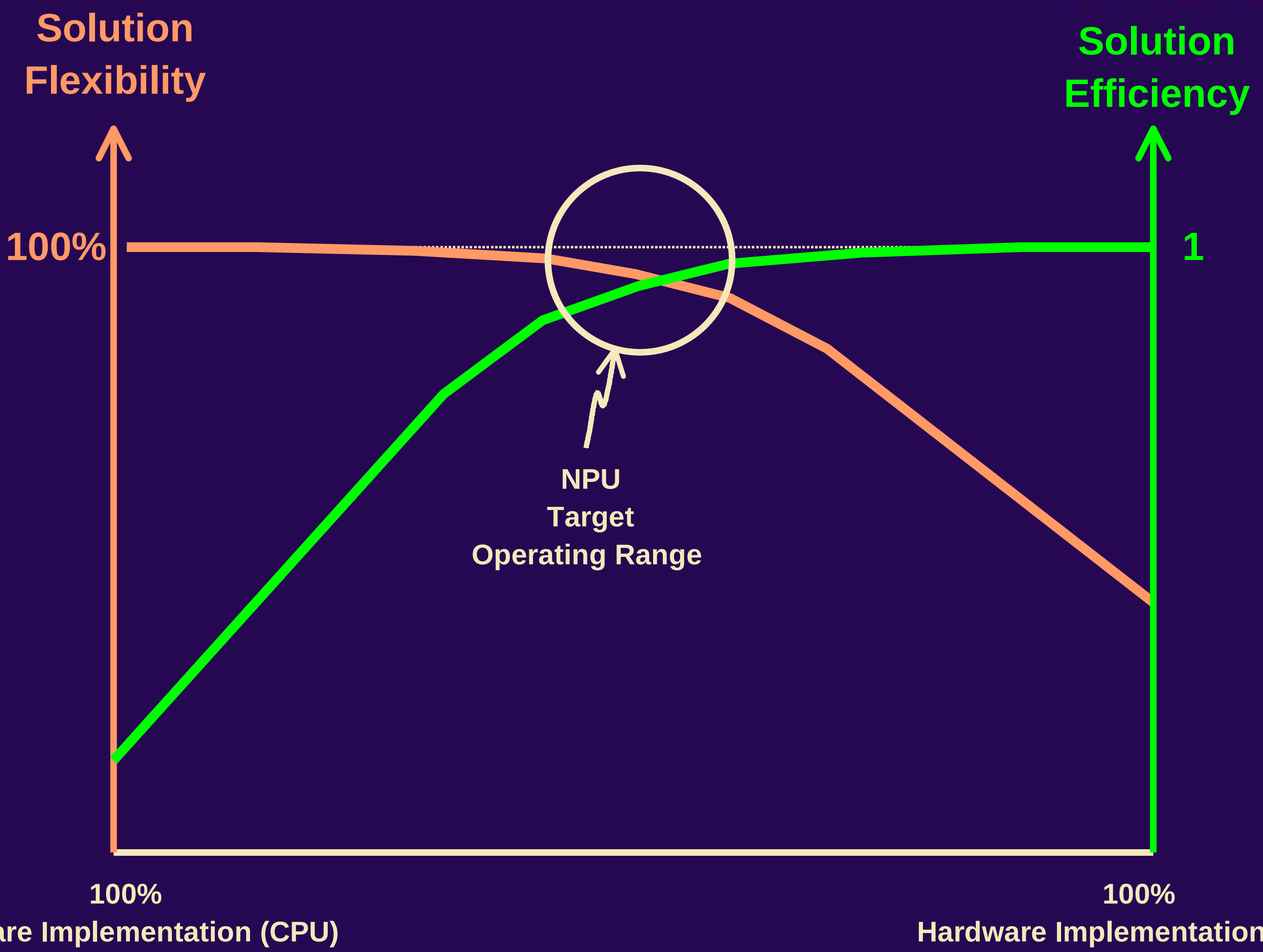
- Same program image loaded on each thread
- A packet runs to completion on a single thread



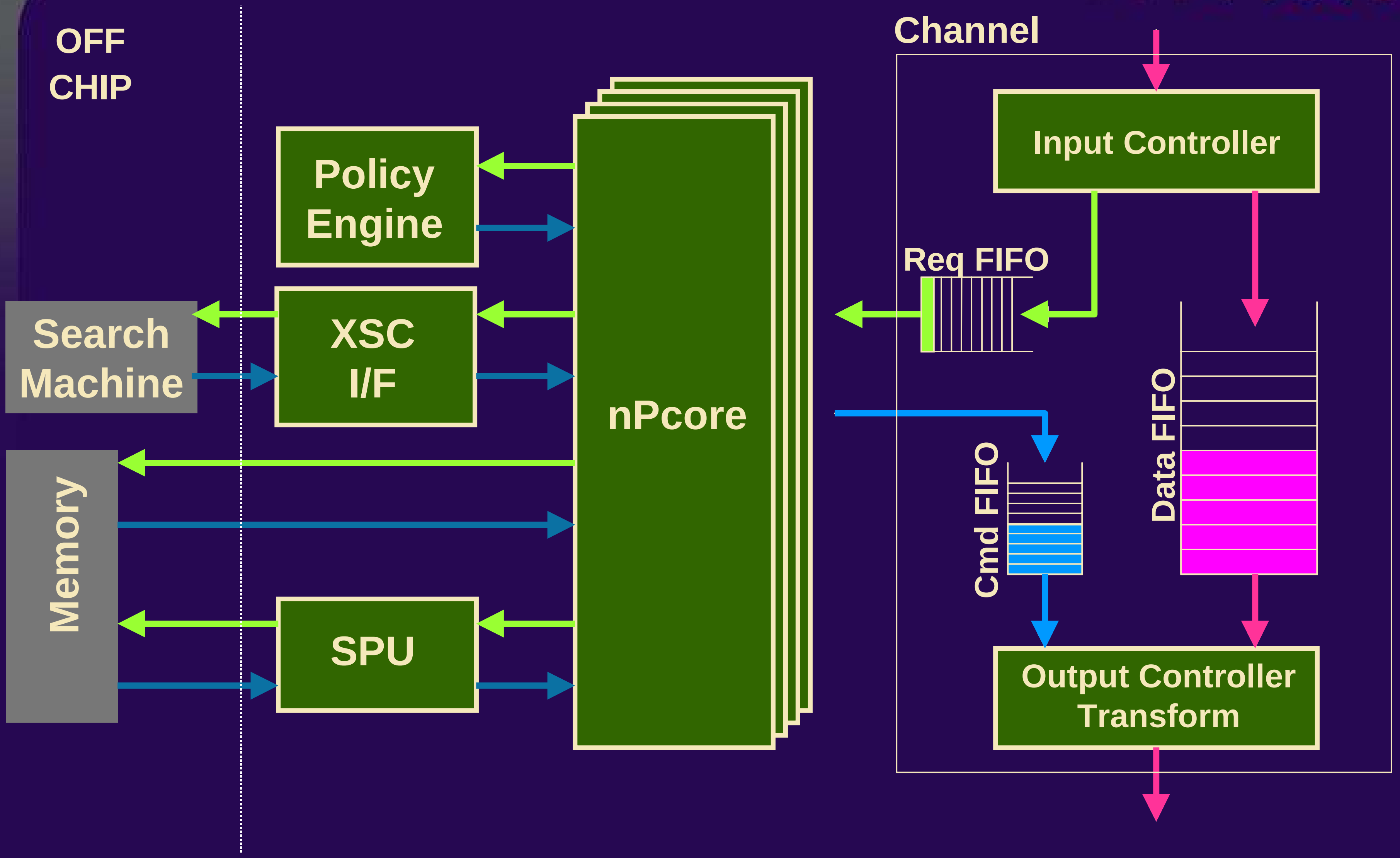
```
;;    cannot handle.
;;
;;    An encapsulation header is used to communicate the
;;    source-port-id and the errno value as follows:
;;
;;                      +====+====+====+====+====+====+====+====+
;;                      | _0 | _1 | _2 | _3 | _4 | _5 | _6 | _7 |
;;                      +====+====+====+====+====+====+====+====+
;;                      | 81 | 00 | S-PORT  |  ERRNO  |   N/A   |
;;                      +----+----+----+----+----+----+----+----+
;;
;;=================================================================

pkt_fwd_to_cpu
#define DB_HDL r6
        ;; retrieve the control connection switch header from the
nPkernel
        ;; unicast port table.
        set_entry_db ucast_port_table, DB_HDL, APP_CPU_PORTID
        read_first_db ucast_port_table, DB_HDL, r0
        ;; setup the encap header
```



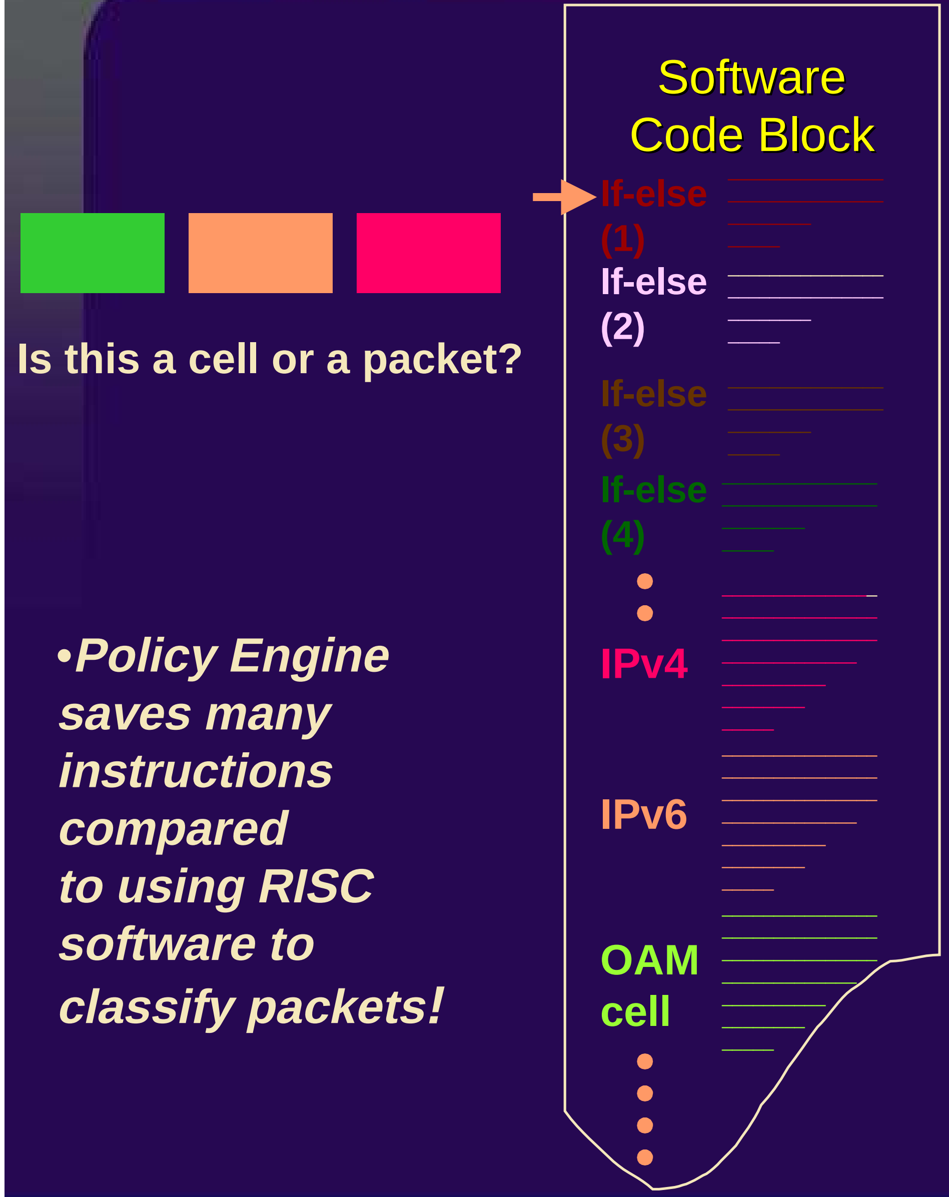
Challenge: Flexibility vs. Performance



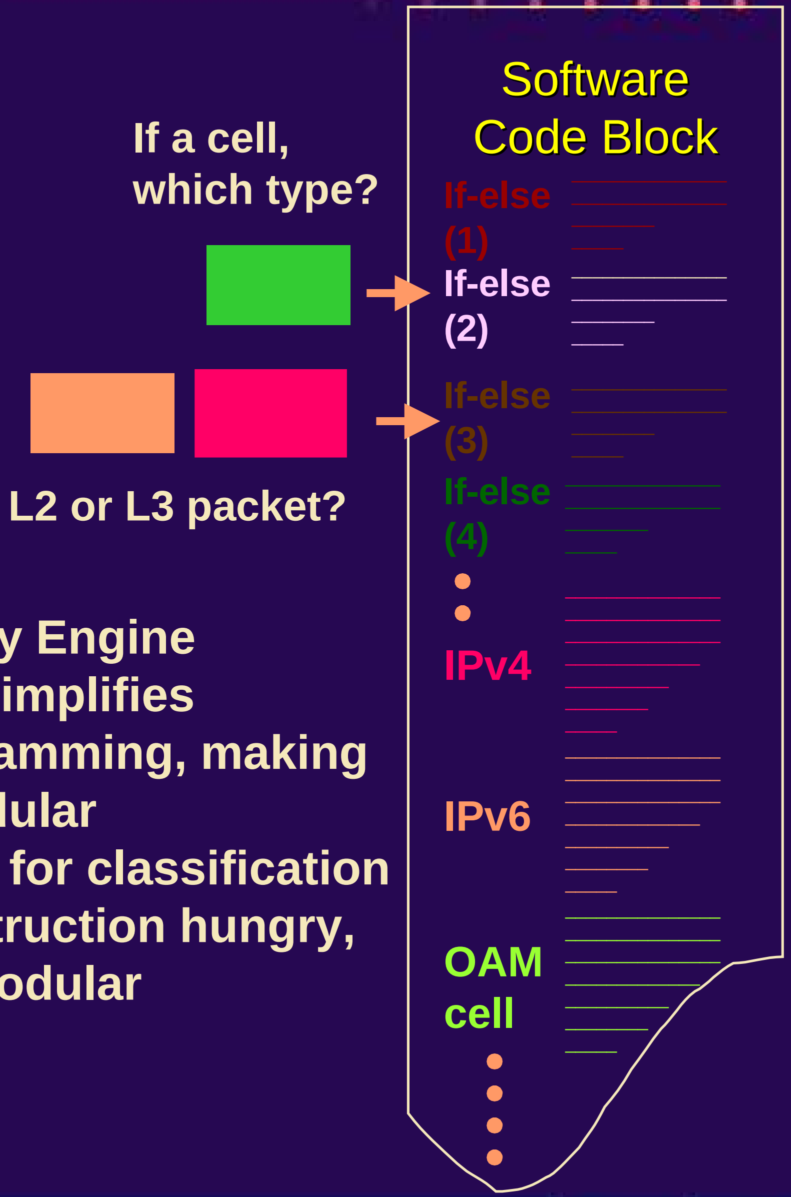
Solution: Co-Processors



High Speed Single-Step Packet Classification



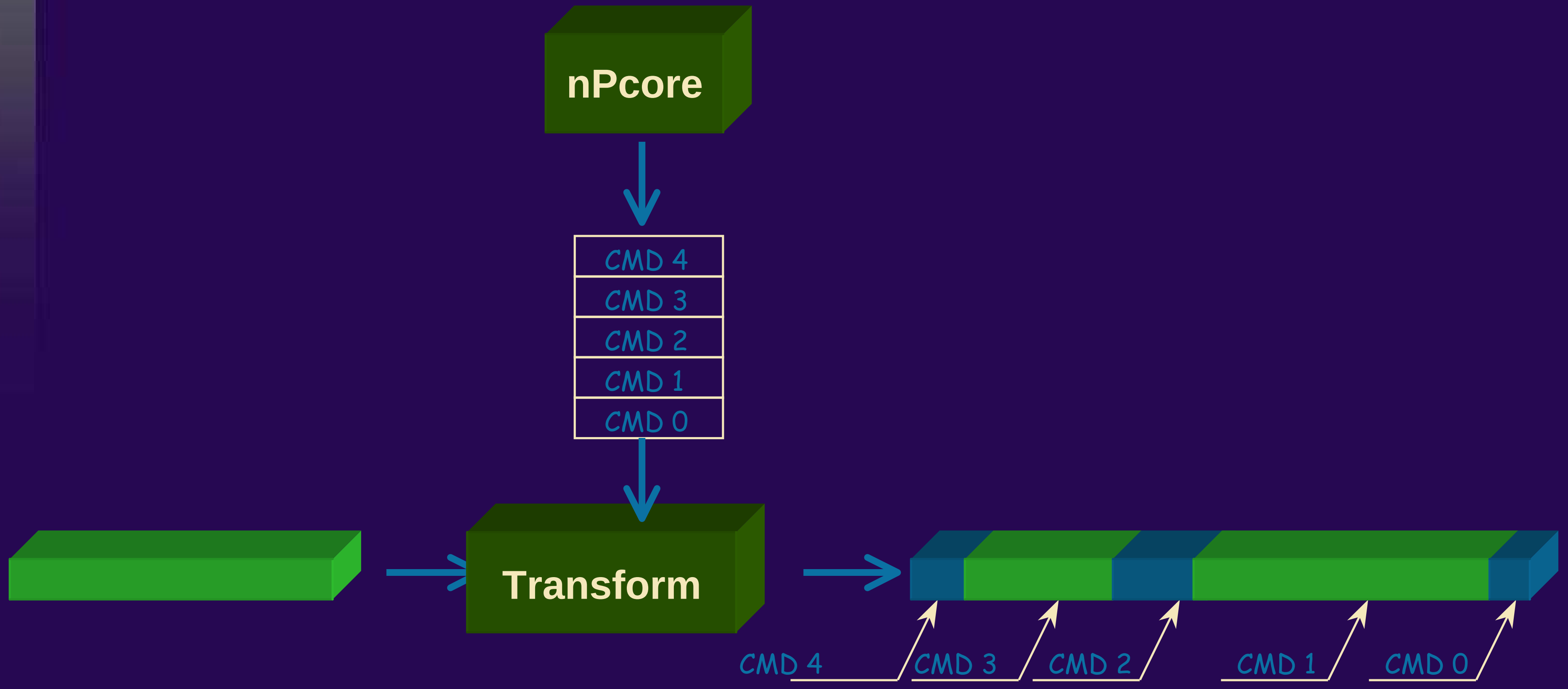
•Policy Engine saves many instructions compared to using RISC software to classify packets!



•Policy Engine also simplifies programming, making it modular

•RISC for classification is instruction hungry, not modular

Efficient Packet Modification & Encapsulation



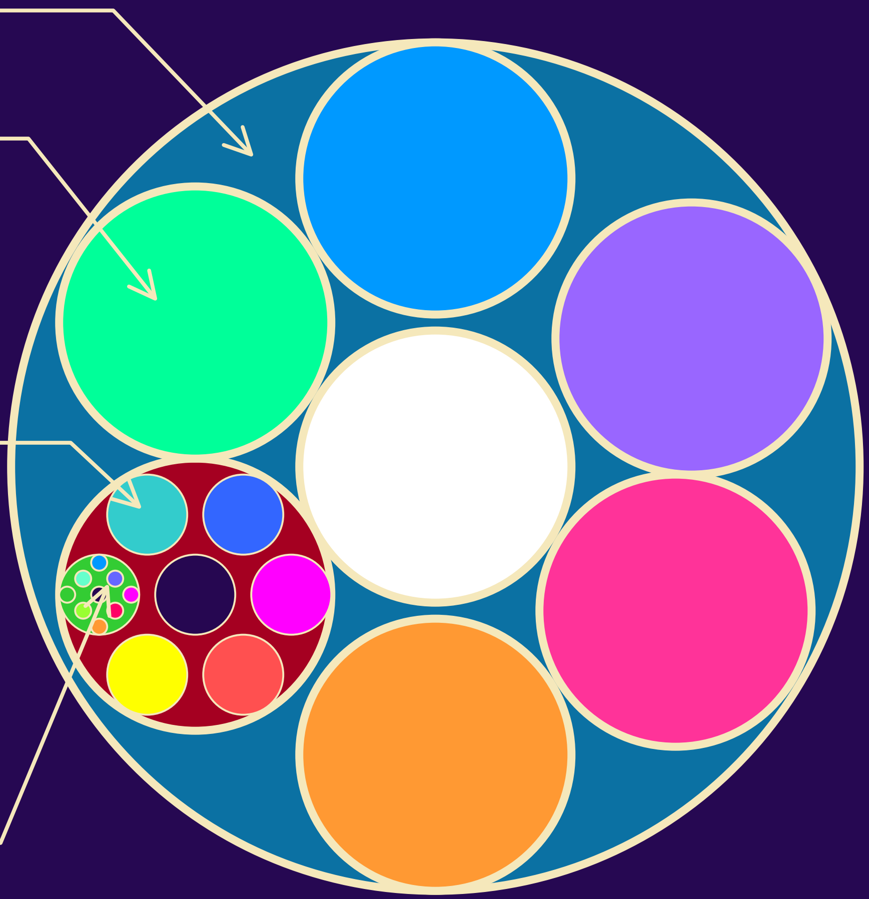
Complex Bandwidth Provisioning

PHYSICAL PORT
• OC-192

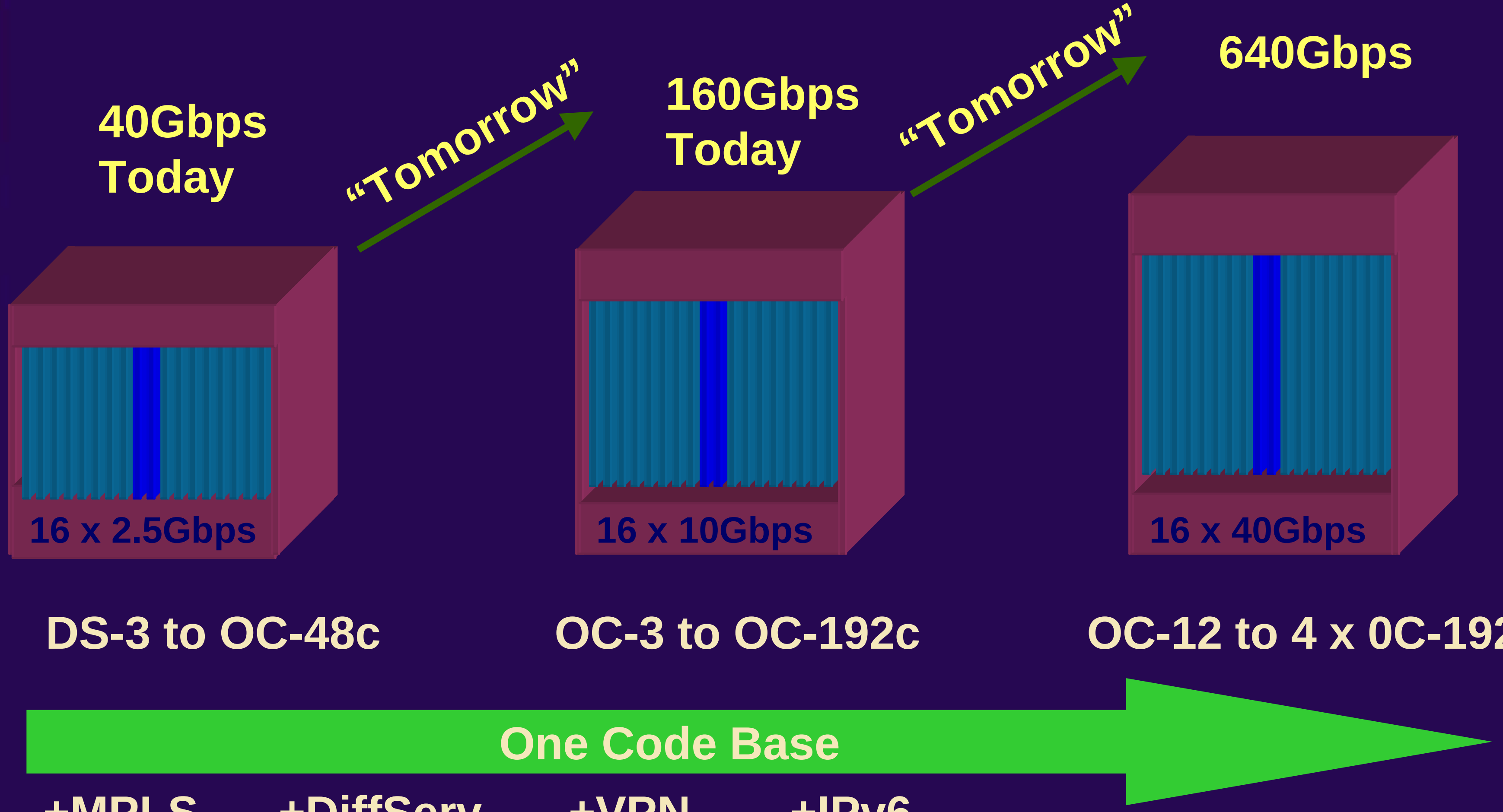
SUB-PORTS
• Fixed sub-division of the physical port
• OC-192, -48, -12, -3, GE
• Flows are scheduled according to minimum rate, class of service and weight at this level

VIRTUAL PIPES
• A collection of flows that have an aggregate maximum rate
• Pipe ensures that provisioned rate is not exceeded
• For example - all flows for an individual subscriber, network or traffic type

FLOWS
• Smallest scheduled entity
• May have guaranteed minimum bandwidth
• Individually weighted within a pipe



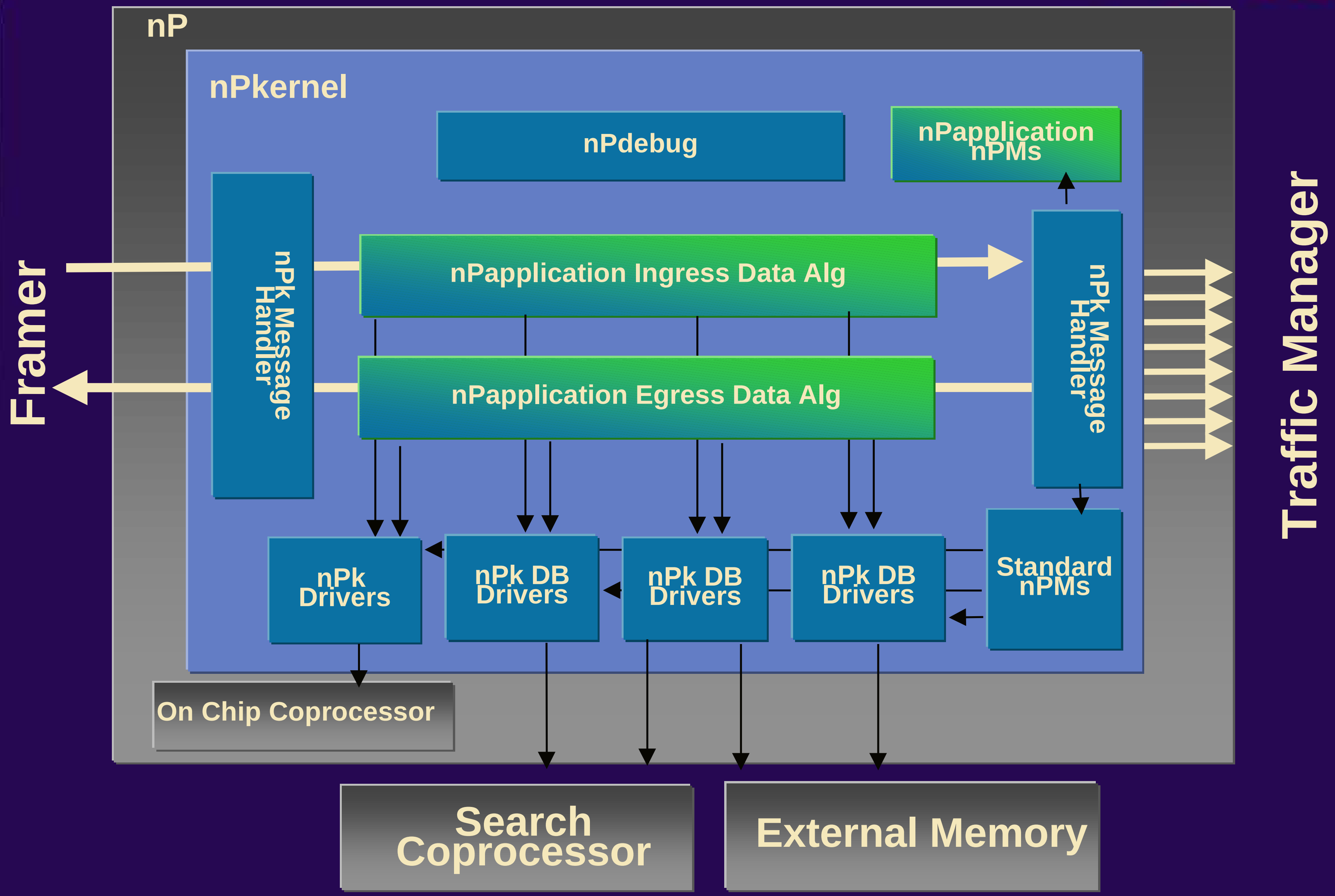
Challenge: Scalability & Re-use



+MPLS +DiffServ +VPN +IPv6

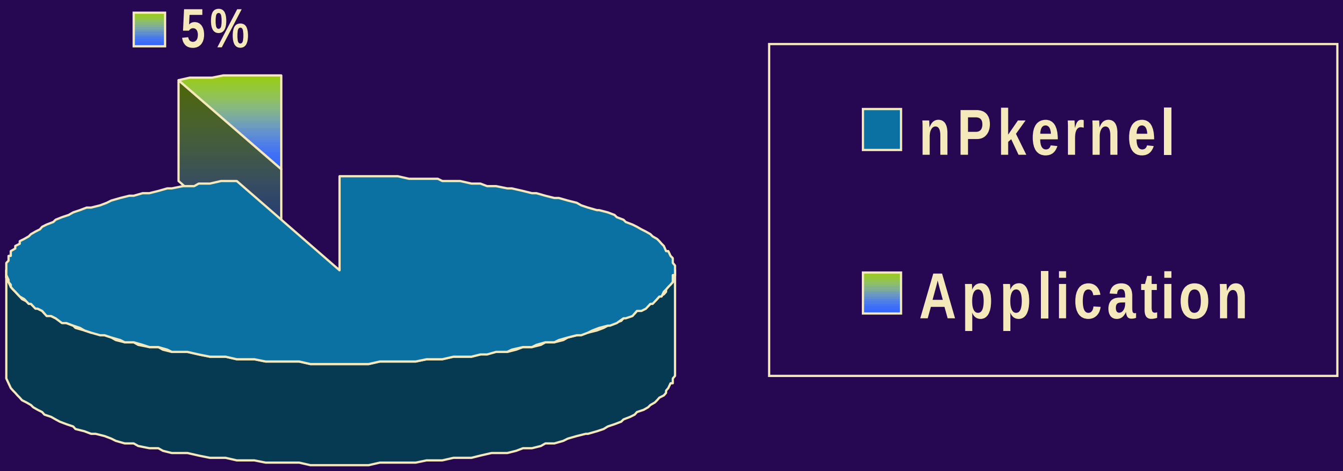
Maximum Software ROI

Solution: nP Programming Infrastructure

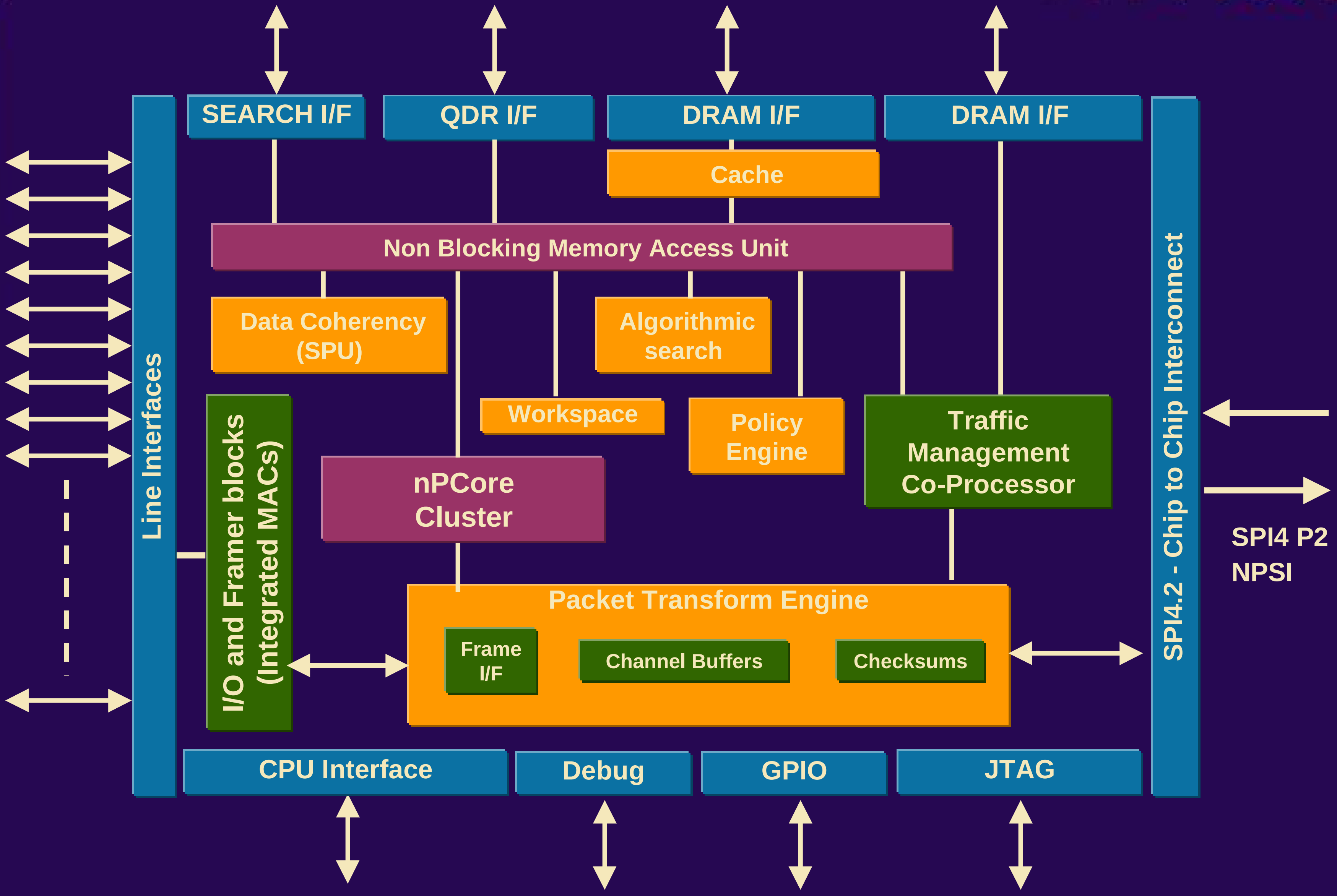


“Runtime Libraries”: Reducing code customer must actually write

- 99+% of total Lines of Code (LOC) are on the control CPU
- Next, nPkernel alone provides up to 95% of actual NPU code, i.e. 95% of the 1%
- So NPU-resident portion of app often only 100-200 total LOC
- AMCC sample app u-code provides most of that 100-200 LOC, all in some cases
- Actual customer-generated code typically much less than 5% of 1% !!



Putting it all together - Integrated NPU



Summary

- NPU Definition – No performance penalty
- Challenges & Solutions
  - Physical – More MHz won’t cut it
  - Performance – Cannot be at the price of usability
  - Flexibility – Doesn’t mean 100% software
  - Scalability & Re-use – Any Protocol, Any Speed, One Architecture
- Putting it all together
  - Avoiding System Problems & Side Effects