VaST SYSTEMS TECHNOLOGY CORPORATION

Empirical Architecture:

Balancing Computation and Communication

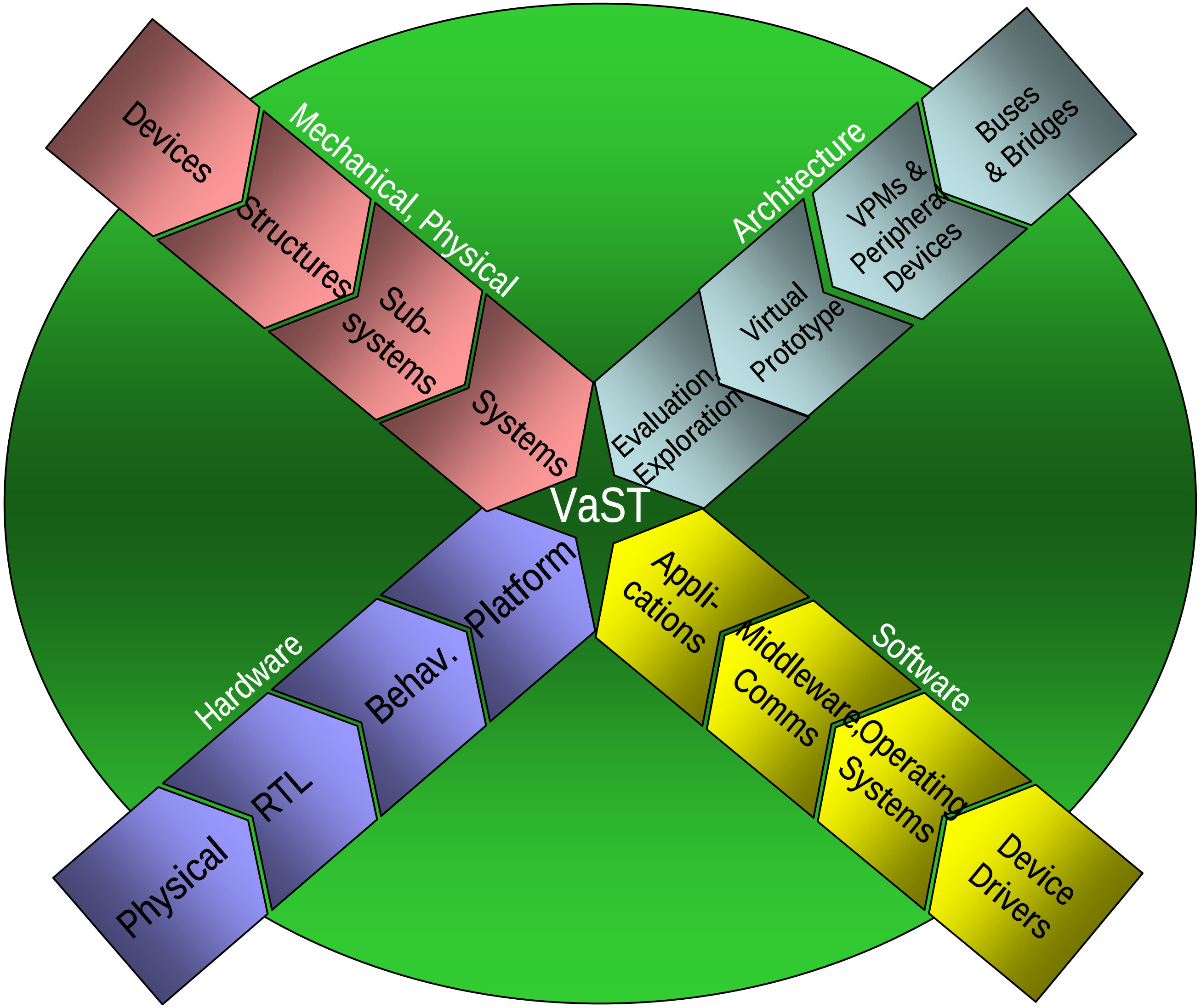
Graham Hellestrand, Ahmed Abdallah

MPSoC Workshop, Estes Park, Co, 15 Aug 2006

Systems Architecture Scope

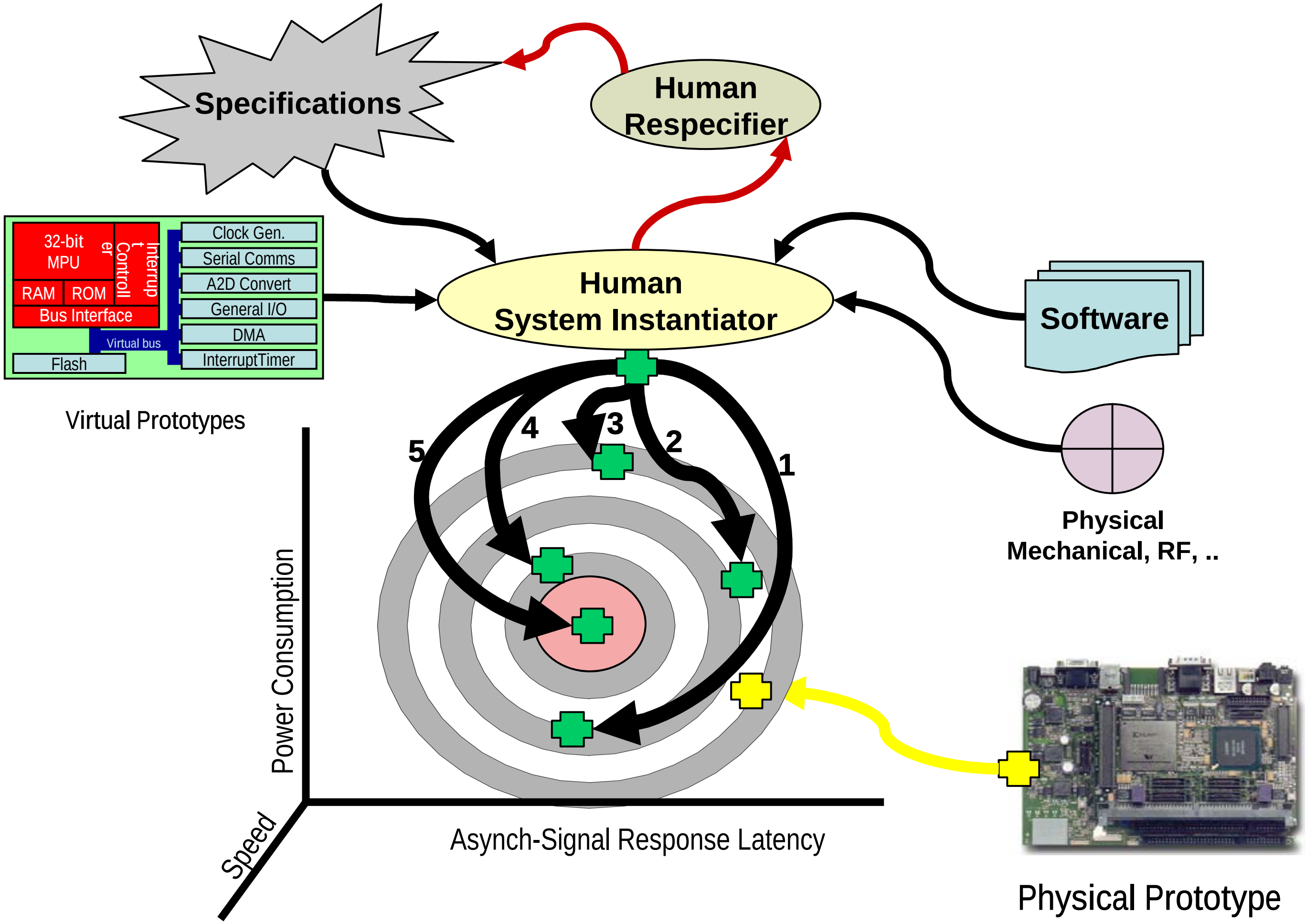
Must be Pre-Silicon!

Architecture Incorporates the Whole System



Empirical Architecture:

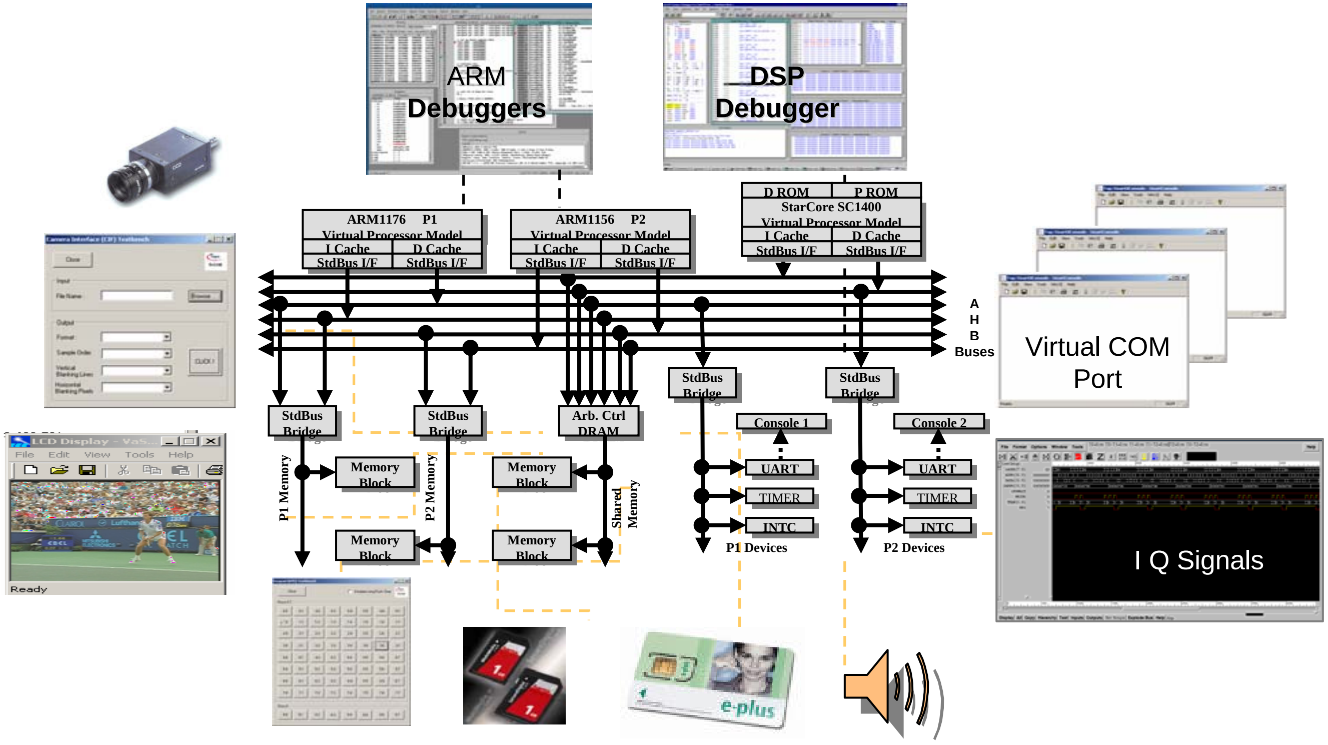
Its about Optimal System Specification using Models



Architecture
&
The Challenges of Modeling

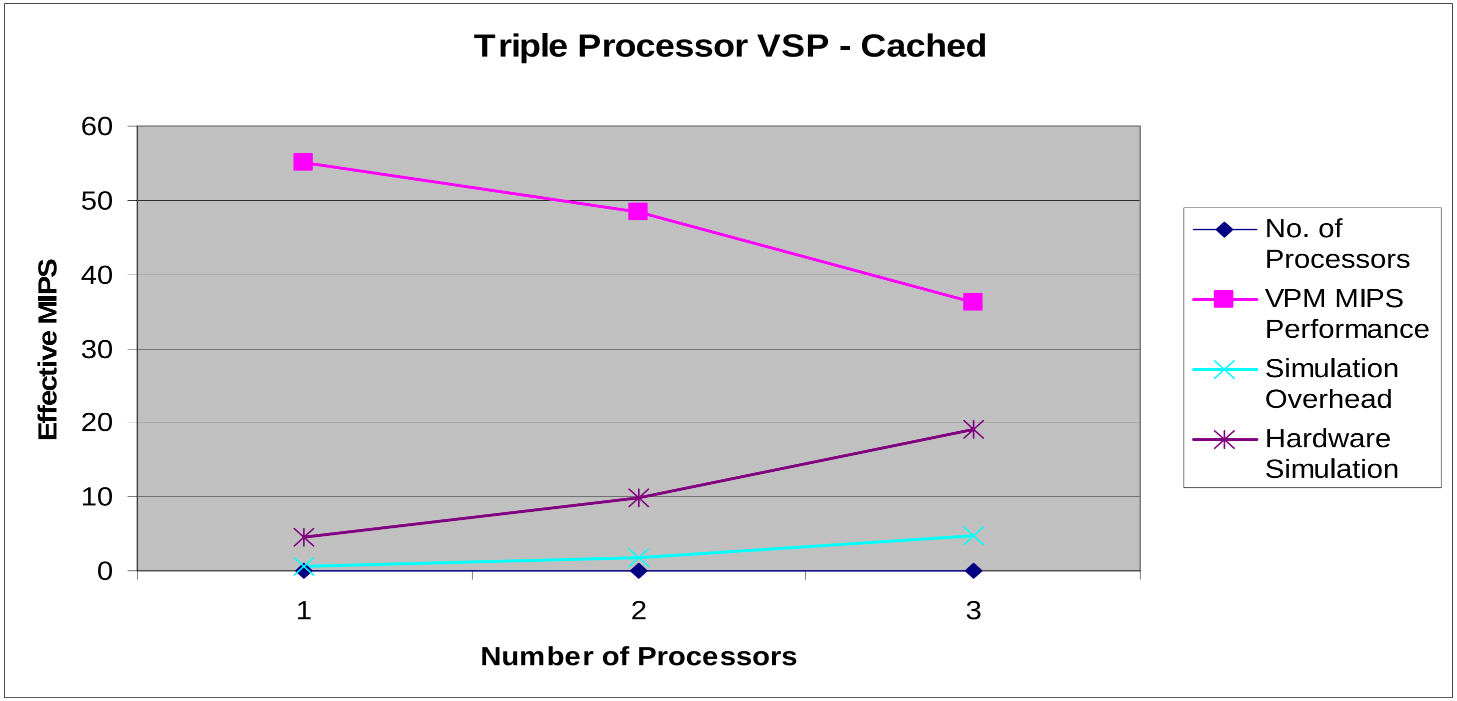
A Paradigm Industry System:
Virtual System Prototype (VSP) of a 3G-Mobile Handset Development

- Full System Simulation & Debugging
  - 3 processors, 80 peripheral devices
  - Typical performance 20+ MIP



MP Model Performance on Single Host

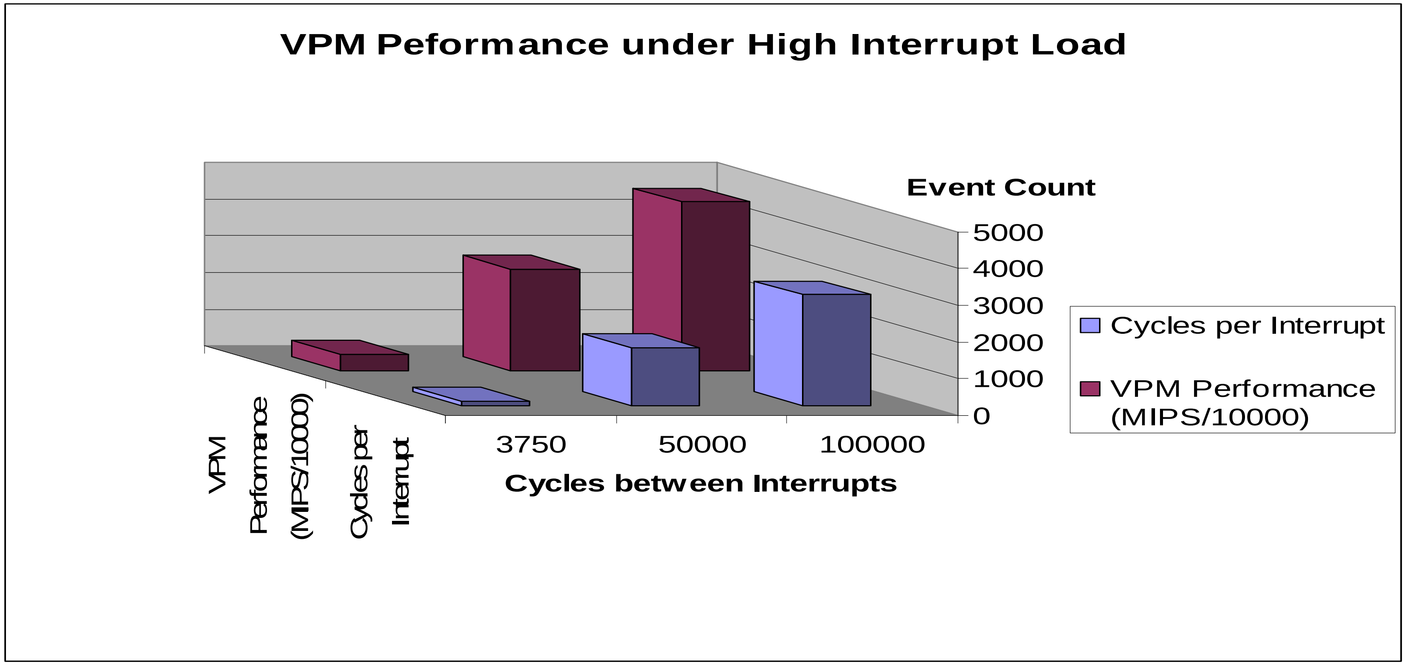
3x Closely Coupled (μP + I&D Cache) Models



Processor dominated study: As processors (VPMs), having software and data resident in cache, together with their subsystems (buses, bus bridges, memory, timers, etc.) are switched into the simulation (Pink line), more hardware becomes active (purple line) so host cycles are shared between the active VPMs and the hardware, as expected. Since switching between processor models and hardware also increases the Simulation overhead (blue line) also increases, but relatively slowly.

SP Model Performance on a Single Host
Single μP VSP Interrupt Handling

Automotive Benchmark, Feb 2004



Capability or a VSP under interrupt loads: This is a relatively simple experiment that shows the performance of a single processor Virtual System Prototype under increasingly stressful rates of processing asynchronous events (interrupts). Even at high interrupt rates (every 3,750 cycles is equivalent to a 12 cylinder engine running at 20,000 RPM and producing an interrupt every 10 degrees of crank-angle) the VPM is capable of simulating high software execution rates while handling such interrupt frequencies.

Modeling Conclusion

A single host simulation environment is adequate for
architecture and
software development
for Functionally complete, timing accurate
Virtual System Prototypes (VSP)
containing
6-10 closely coupled processors in a VSP
or
6-10 distributed VSPs

MP Architectures:

Balancing Computation and Communication

Balancing Systems: Computation – Communications

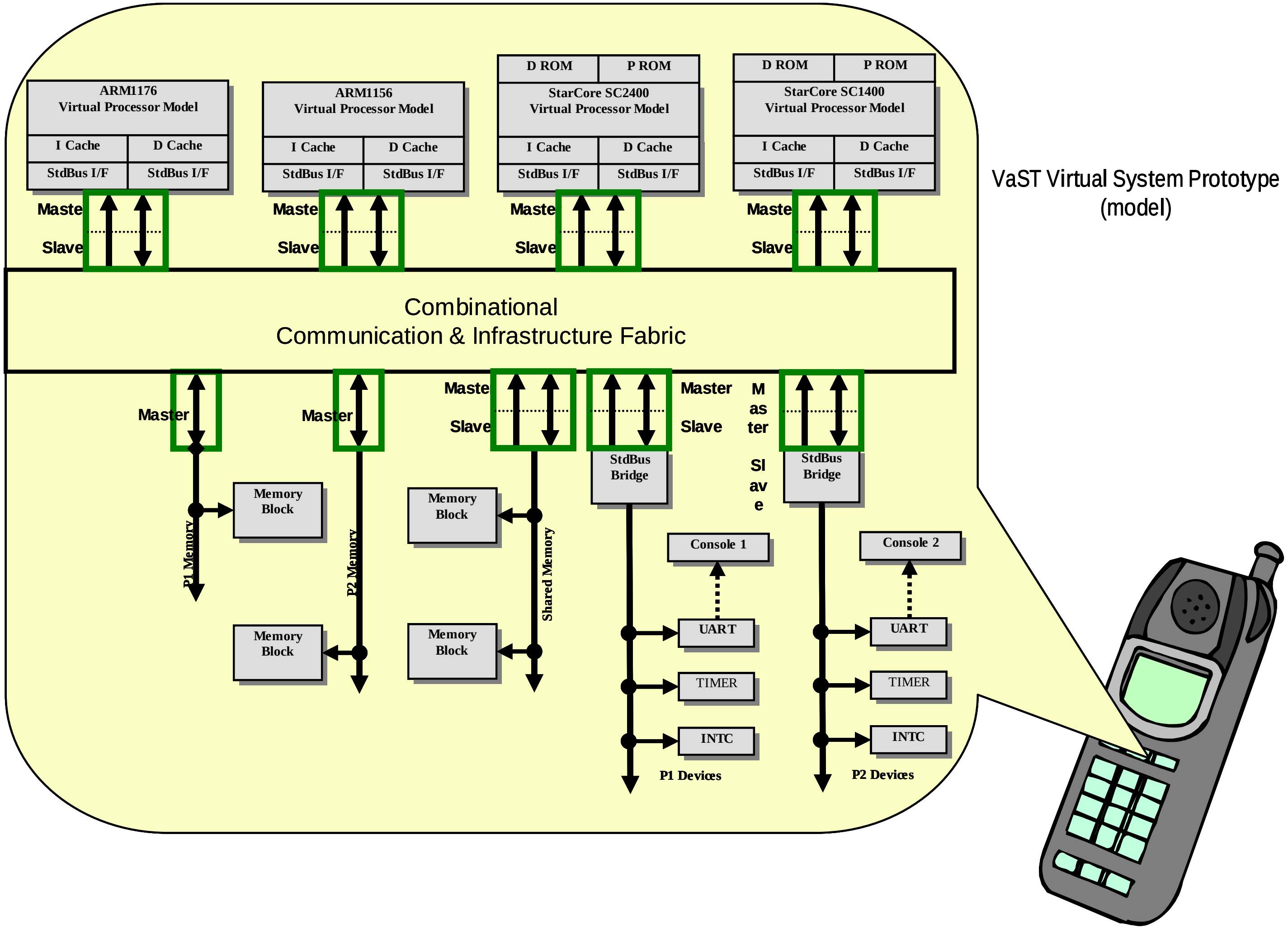
Type 1: Cooperative Computation

- Closely-Coupled Processor configurations
  - Heterogeneous – more general
  - Homogeneous – easier to schedule + SMP

- Balanced Communications Requirements
  - Low communications latency
  - High bandwidth

Platform Architecture

1st Order Impact



VaST Virtual System Prototype
(model)

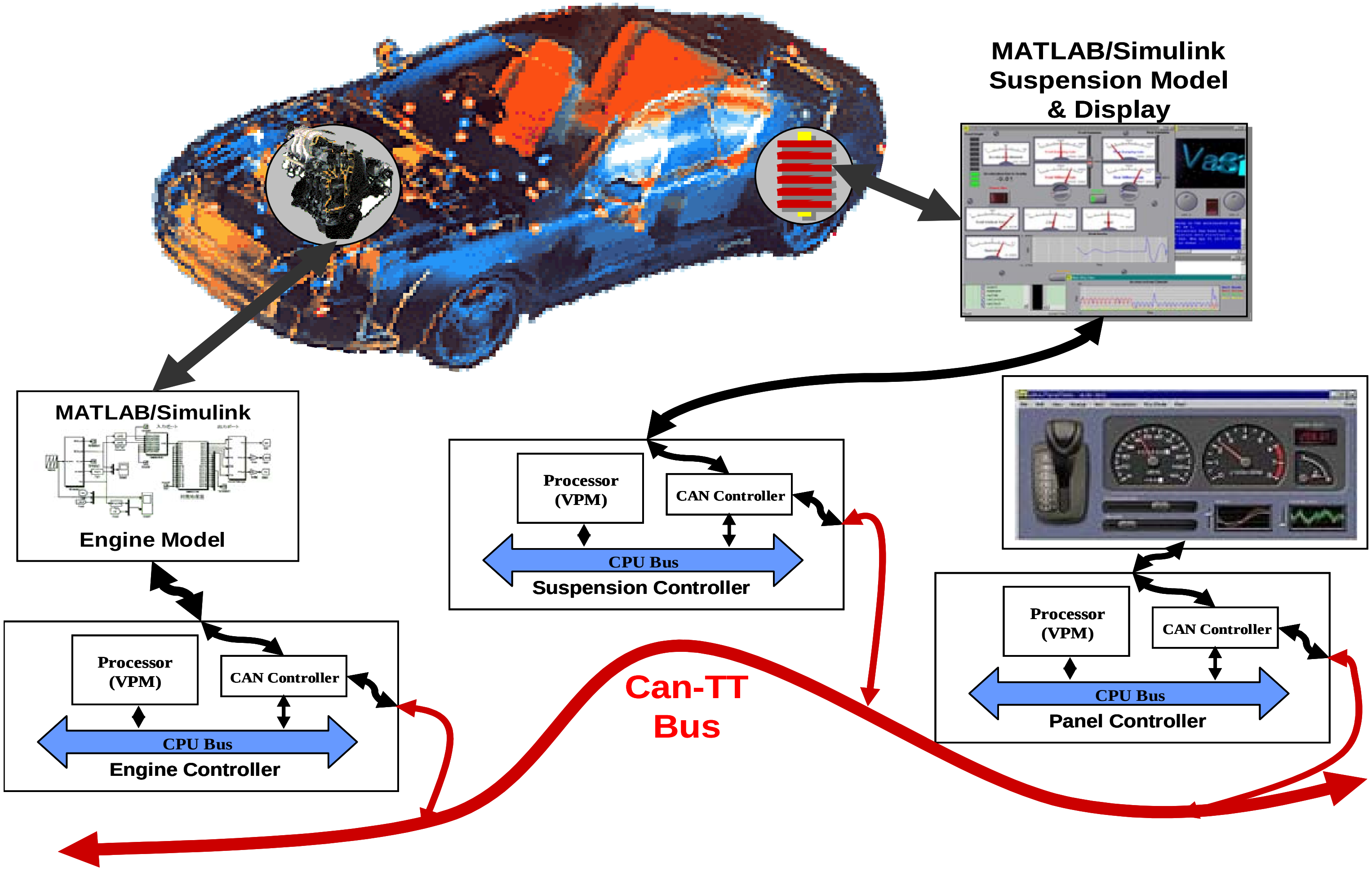
Balancing Systems: Computation – Communications

Type 2: Distributed, Local Computation

- Local, Separate Processing subsystems
  - Typically heterogeneous
- Communications Requirements
  - Tolerates high communications latency
  - Low bandwidth

Distributed Systems Architectures

Typically Heterogeneous – Coarse Grained Concurrency



MATLAB/Simulink Suspension Model & Display

MATLAB/Simulink

Engine Model

Processor (VPM)

CAN Controller

CPU Bus

Suspension Controller

Processor (VPM)

CAN Controller

CPU Bus

Panel Controller

Processor (VPM)

CAN Controller

CPU Bus

Engine Controller

Can-TT Bus

Balancing Systems:

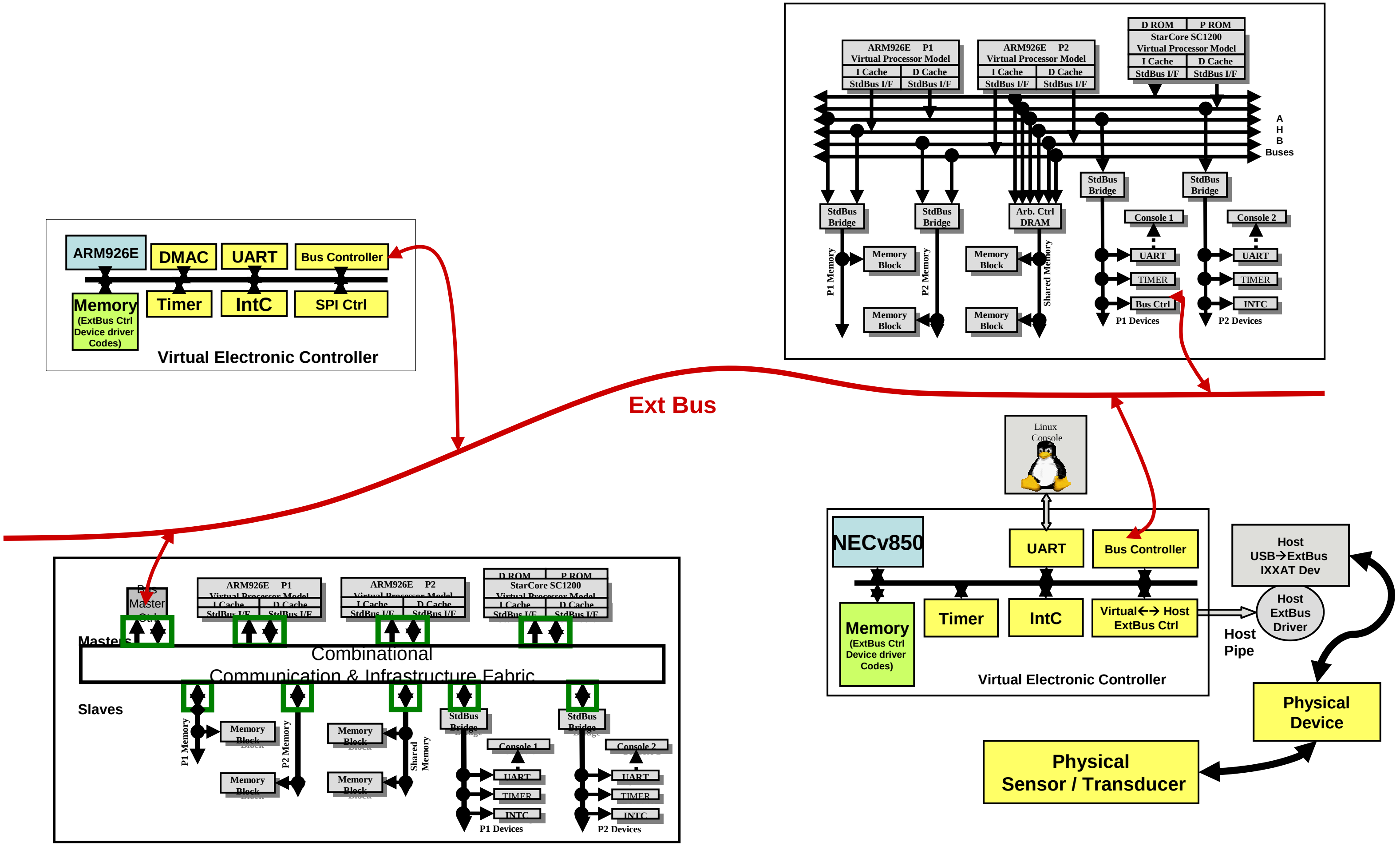
Type 3: Hybrid: Mapping into the Computation–Communications Space

- Monolithic, Cooperative Computation with high interdependency
  - Computation: Closely-coupled computation engine
  - Communications: Low latency, High bandwidth
- Local Computation with Casual Interaction
  - Computation: Separate computation engines
  - Communication: High latency, Low bandwidth
- All points in computation – communication space between

Hybrid System:

Networked, Real-time Control and Multi-Media Application

(Mapping fine and coarse grained concurrency)



Measuring Architectures and Interpreting the Measure Meanings

Measuring, Simplifying, Interpreting

- **Measuring the VSP**
  - Platform
    - + Relatively static model of hardware architecture
    - + Capabilities (eg D-cache) can be manipulated dynamically – usually once
  - Software
    - + Static architecture with an execution trace that is data dependent
    - + Directly stimulates the platform
  - Data
    - + Primarily consumed by software and resulting stimulus *drives* the platform

  Difficult to interpret (recognize patterns in) millions of event measurements of hundreds of simulation variables extracted from the VSP - say executing 300 million instructions - during an experiment.

- **Need to reduce the number of variables that characterize the VSP**
  - Compute the correlations between variables (R-factor analysis – where factors are linear combinations of observed variables)
  - Find the set of variables (called a factor) with variance that accounts for the highest common variance amongst the set of variables
  - Continue this process, excluding the already determined sets (factors), until no group exists that is capable of accounting for the variance of at least one variable

  The factors are a smaller set of *latent* (unobservable) variables that can explain the system of variables. Factors are a linear combination of the variables in their defining group; the variable are said to load on a factor according to their correlation with that factor.

- **Cogent interpretation - explain (name) the factors**

  The interpretation is in terms of identifiable, data dictated characteristics of systems derived by summarizing the meanings of the significant loadings of each variable (factor) on the set of factors (variables).

Experiment 1: Summary Data

Booting MVL v2.1 Linux on ARM926E based Platform (simple memory hierarchy)

100 million Cycles before End of Decompresion of Linux Image (Sampled Data ~100 million cycles)

Representative Variables	Rotated Factors	1 Logical Computation	2 Data Move Control	3 Data Storage	4 UART Device Service	5 Load Multiple	6 Store Multiple	7 Instr - Memory Hierarchy
	Total Common Variance	**11**	**5.9**	**3.3**	**2.4**	**2.4**	**1.6**	**1**
	Correlations	**1-3 (.28)**	**2-7 (-.25)**	**1-3 (.28)**				**2-7 (-.25)**
Positive Correlations >0.5	**Instr Class**	Logical Compar Load	Move Compar	Store		Load multiple	Store multiple	
	Instr Class Stalls	Logical Compar Load (.46)	Move Load Branch	Store (.45)		Load multiple	Store multiple	
	Data-Path Registers	Access		Access				
	Cache-Data	Read miss		Write miss				
	Cache-Instr							
	MemHier-Data	Read	Write delay	Write Write delay				
	MemHier-Instr							
	Device				Read Write Tx Data			
	TLB							
	Exceptions							
	CoProc (CP15)	Access (.31)	Access (.47)	Access (.47)				Access
Negative Correlations < -0.5 (Not …)	**Instr Class**	Arith Branch	Logical	Logical				
	Instr Class Stalls	Arith	Logical Compar					Store (-.4)
	Data-Path Registers		Reg. Access (-.44)					
	Cache-Data							
	Cache-Instr							
	MemHier-Data							
	MemHier-Instr	Fetch Fetch Delay	Fetch delay (-.39)					Write delay (-.47)
	Device							
	TLB							
	CoProce (CP15)							

Experiment 1 – Interpretation

~100 million Cycle Sample during Linux Image Decompression

- Evidence:
  - 3 Dominant Factors (~80% of correlation)
    - + Logical Computation (in-line) + Data Storage
    - + Data Move Control
  - 2 Notable Ancillary Factors
    - + Output mssgs from Linux boot via UART device
    - + Infrequent structured data movement
      - – likely function calls

- Interpretation of Behaviour:
  - Experiment sample is dominated by the decompression algorithm
  - Other work
    - + UART service outputs messages

Experiment 2: Summary Data

Booting MVL v2.1 Linux on ARM926E based Platform (simple memory hierarchy)

After Decompression - from Start of Boot (Sampled Data ~100 million cycles)

Representative Variables	Rotated Factors	1 Structured call (function)	2 Move data	3 Structured return (function)	4 Branch Predict	5 VM Operation	6 Device Setup & Service	7 Data store Memory Hierarchy	8 Data Load Mem. Hierarchy	9 OS services invoked	10 Logical / Bit Manipulation	11 Control Unit set-up & Actions
	Total Common Variance	6.8	6.6	5.2	4.5	4.2	4.1	3.95	3.9	3.8	3.7	2.1
	Correlations	1-3 (.37)		1-3 (.37), 3-7 (.27)		5-9 (.3)		3-7 (.27)		5-9 (.3)		
Positive Correlations >0.5	**Instr Class**	Store multiple	Move Load multiple Store multiple	Load multiple	Compar Load			Store	Move (.47) Load		Logical	
	Instr Class Stalls	Store multiple Branch (.35)	Move Logical	Load multiple Move (.31)	Load Branch			Store	Compare		Logical	
	Data-Path Registers								Access (.39)		Access	
	Cache-Data	Write miss	Write hit Read miss	Line evict	Read hit (.46)	Line evict Line fill (.41)		Line Evict (.36) Line Write-back	Read hit Write hit			
	Cache-Instr					Miss Line evict Line fill			Hit (.32)	Miss (.35) Line evict Line Fill (.35)	Hit (.47)	
	MemHier-Data	Write Write delay		Read				Read (.37)				
	MemHier-Instr									Fetch Fetch delay		
	Device		Read Write Tx Data				Dev. Reg Access Int. Enable Write Dev Status Read					
	TLB					Miss						
	Exceptions						Int.Request to µP			Swi		
	CoProc (CP15)											Access Access stall
Negative Correlations < -0.5 (Not …)	**Instr Class**	Branch (-.4)		Branch (-.31)	Arith				Logical (-.38) Branch (-.33)		Branch	
	Instr Class Stalls		Branch		Arith				Logical (-.34)		Arith	
	Data-Path Registers											
	Cache-Data											
	Cache-Instr	Hit										
	MemHier-Data											
	MemHier-Instr											
	Device											
	TLB											
	Exceptions											
	CoProce (CP15)											

Experiment 2 – Interpretation

~100 million Cycle Sample – immediately after decompression

This is a much more complex analysis with more spread clustering of activities. However is still clearly the trace of an operating system booting.

- Evidence:
  - 3 Large Factors
    - + Structure Call/Return and Moving Data (26% of variance)
  - 5 Typical OS initiation activities
    - + Extensive data and device set-up
    - + Data load/store memory hierarchy with VM
    - + Control unit (CP15) initialization
    - + Installed OS service invocations
    - + Servicing of device interrupts
  - Architecture Improvement Indicator
    - + Branch prediction indicated
- Interpretation on Linux Behaviour:
  - Installing the kernel, I/O services, process scheduling services, device drivers, etc.
  - Control unit initialized for I&D cache and VM
  - VM, OS and device services enabled
  - The correlation between Comparison instructions and Branch stalls indicates a potential efficiency gain by implementing simple branch prediction
  - The 3 main factors comment on the consistent structuring of Linux in execution

Conclusions on Measurement

Factor Analyzing MVL v2.1 Linux Booting on a ARM926E, Simple Platform

- After preliminary experimentation:
  - Separated experiments into 3 defining parts:
    - Expt 1: Decompressing Linux image
    - Expt 2: Start of boot
    - Expt 3: Near Idle process
- ~50 variables measured, for example:
  - I & D cache – hit, miss (read, write), fill, evict
  - Instruction classes
  - Stall times
  - Etc.
- Factor analysis:
  - Reduced the information to be considered from ~50 to between 3 and 8 main factors
  - For each experiment:
    - Factors have a reasonably good physical meaning
    - The explanation of a complex OS in its main boot sequence (100 million instructions) was understandable
  - Factor analysis indicated the potential efficacy of simple branch prediction for the ARM926 platform

Optimizing Architectures for Power and Performance

Separated Functions

General Form of Multi-Objective Optimization Equation:

Characterize an objective function in terms of events directly measurable from the VSP

$$\boxed{\begin{aligned}
F_{VSP}(| \; & f_{CPU}(\Theta_{cc=0..cn} \circ f_{CPU_{cc}}(\Theta_{CEvType=1..cet} \circ g_{CPU_{cc,CEvType}}(\Theta_{CEvCnt=scecn..tcecn} \circ Event_{CPU_{cc,CEvType,CEvCnt}})), \\
& f_{Bus}(\Theta_{bc=0..bcn} \circ f_{Bus_{bc}}(\Theta_{BEvType=1..bet} \circ g_{Bus_{bc,BEvType}}(\Theta_{BEvCnt=sbecn..tbecn} \circ Event_{Bus_{bc,BEvType,BEvCnt}})), \\
& f_{BusBridge}(\Theta_{bbc=0..bbcn} \circ f_{BBus_{bbc}}(\Theta_{BBEvType=1..bbet} \circ g_{BBus_{bbc,BBEvType}}(\Theta_{BBEvCnt=sbbecn..tbbecn} \circ Event_{BBus_{bc,BBEvType,BBEvCnt}})), \\
& f_{Mem}(\Theta_{mc=0..mcn} \circ f_{Mem_{mc}}(\Theta_{MEvType=1..met} \circ g_{Mem_{mc,MEvType}}(\Theta_{MEvCnt=smecn..tmecn} \circ Event_{Mem_{mc,MEvType,MEvCnt}})), \\
& f_{Dev}(\Theta_{dc=0..cn} \circ f_{Dev_{dc}}(\Theta_{DEvType=1..\mathrm{det}} \circ g_{Dev_{dc,DEvType}}(\Theta_{DEvCnt=sdecn..tdecn} \circ Event_{Dev_{dc,DEvType,DEvCnt}})))
\end{aligned}}$$

$$where \;\; f_{CPU_k}(\Theta_{EvType=1..et} \circ g_{CPUk,EvType}(...)) = f_{CPU_k}(g_{CPU_k,1}(...), g_{CPU_k,2}(...) \,,....., g_{CPU_k,et}(...))$$

Problem: Huge volume of data some of which may be highly correlated with other data – leading to multiple counting and unreliability in composite measures.

A Simple Power Function for a Full Platform

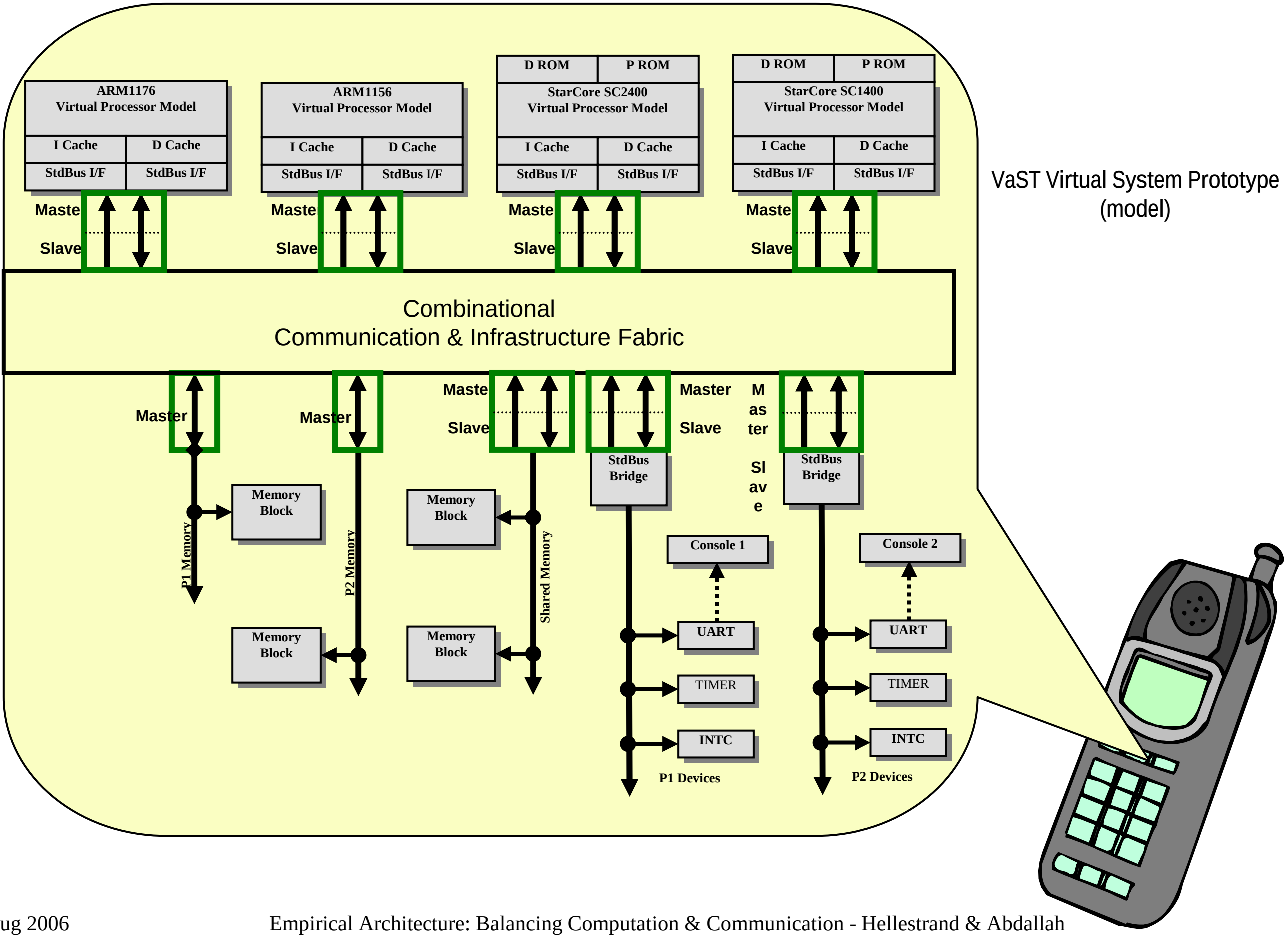
$$
\begin{aligned}
&\textit{Equation 2}:\\
&f_{Power} = .W_{Pipe} \times f_{Pipe} + W_{Instr} \times f_{Instr} + W_{Cache} \times f_{Cache} + W_{TLB} \times f_{TLB} + \\
&\qquad W_{\mathrm{Re}\,gAcc} \times f_{\mathrm{Re}\,gAcc} + W_{MemAcc} \times f_{MemAcc} + W_{PeriphAcc} \times f_{PeriphAcc}\\
&\quad where.\\
&\quad f_{Instr} = .2 \times f_{Instr,\,jmp} + 2 \times f_{Instr,except} + 0 \times f_{Instr,ctrl} + 12 \times f_{Instr,coproc_{15}} + \\
&\qquad 0 \times f_{Instr,LdSt} + f_{Instr,arith} + f_{Instr,other}\\
&\qquad and.\\
&f_{Instr,i} = .\sum (instructions\ of\ \ type_i\ \ in\ \ k - cycles)
\end{aligned}
$$

Resolving the Weights for the Power Function

Table 2: Power: Function Types, Event & Weighting Functions

Function Types	Events	Weight Functions
Pipeline	ibase	6.0
Instruction Types	ijmp	2.0
	iexcept	2.0
	icoproc	12.0
	iarith	1.0
Caches (I&D)	Cache_lookup	$f_{i\text{-}dcache}$(size, ways)
	icache_hit	iCache-lookup + f_{icache}(line size, decode)
	icache_miss	Icache_lookup
	dcache_hit	Dcache_lookup + f_{dcache}(size, ways, line size,)
	dcache_miss	Dcache_lookup
TLB	tlb_miss	30.0
Register	regfile_access	1.0
Memory (incl. bus transactions)	membus_transaction	50.0
Periph Device (incl. bus transactions)	periphbus_reg_access	50.0

Closely-Coupled-Processor Systems: Architectures with Low Latency, High Bandwidth Communication Infrastructure



ARM1176
Virtual Processor Model
I Cache
D Cache
StdBus I/F
StdBus I/F
Maste
Slave

ARM1156
Virtual Processor Model
I Cache
D Cache
StdBus I/F
StdBus I/F
Maste
Slave

D ROM
P ROM
StarCore SC2400
Virtual Processor Model
I Cache
D Cache
StdBus I/F
StdBus I/F
Maste
Slave

D ROM
P ROM
StarCore SC1400
Virtual Processor Model
I Cache
D Cache
StdBus I/F
StdBus I/F
Maste
Slave

VaST Virtual System Prototype
(model)

Combinational
Communication & Infrastructure Fabric

Master
Master
Maste
Slave
Master
Slave
M
as
ter
Sl
av
e

Memory
Block
P1 Memory
Memory
Block

P2 Memory
Memory
Block
Memory
Block

Shared Memory
Memory
Block

StdBus
Bridge
Console 1
UART
TIMER
INTC
P1 Devices

StdBus
Bridge
Console 2
UART
TIMER
INTC
P2 Devices

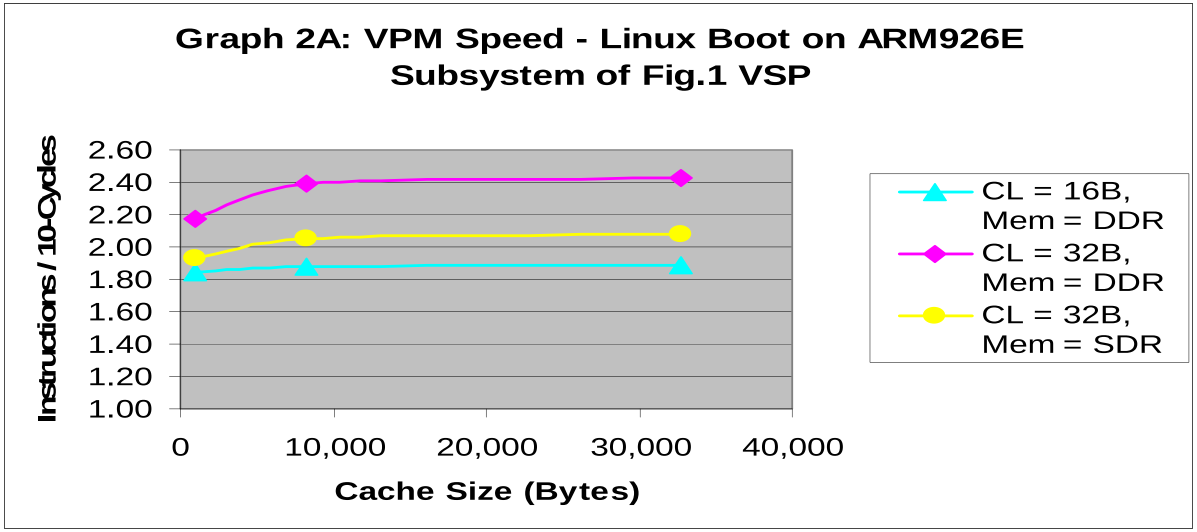
Experiment 3:
Linux Booting under Various Memory Hierarchy Constraints

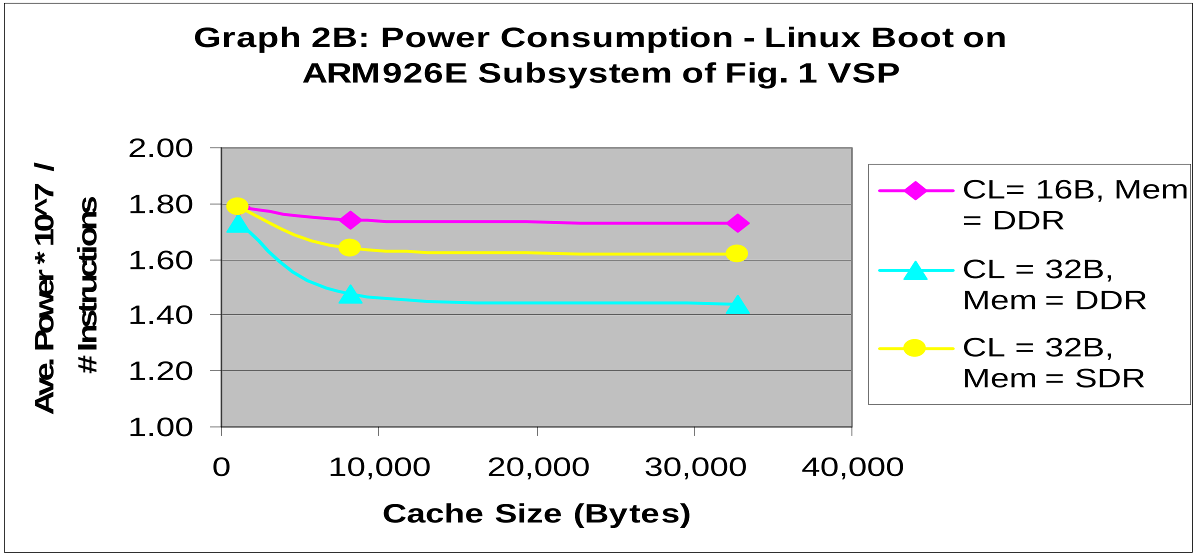
- Objective:
  - To determine the effect of various memory hierarchy configurations (cache size and line size, memory type) in a single processor platform on the performance (instructions / cycle) of, and energy consumed by, Linux booting.
- Experimental Set-up & Results:
  - Linux is booted on various configurations of a single ARM926 processor platform with I & D caches, buses and memory.
  - The results were recorded and displayed graphically.
- The experiments:
  - Linux was booted on various configurations of the platform in which:
    - + I & D Cache line size was set at 16 bytes or 32 bytes
    - + I & D Cache size was increased from: 0 → 32kbytes
    - + I & D memory types were set at Single Data Rate (SDR) and Double Data Rate (DDR)

Experiment 3:

Linux Boot - Memory Hierarchy Analysis

(I&D cache + bus + bus bridge + Mem (DDR | SDR) Analysis





Results of Experiment 3:
Linux Booting under Various Memory Hierarchy Constraints

- During the booting of Linux, the Data cache is enabled about 1/3 of the way through the boot.

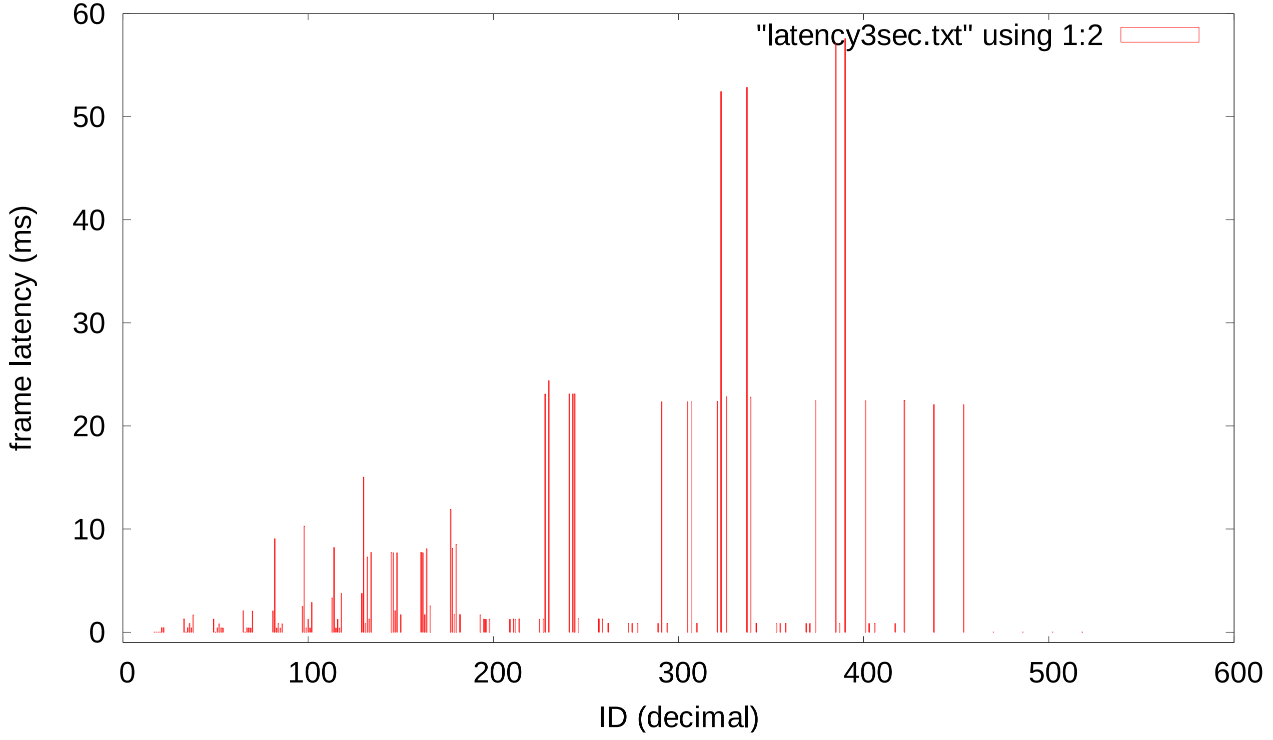
- The performance and power measures are not as dramatically demarked as for the simpler Viterbi program running largely from cache. However, and two major effects are still readily observable.

- When cache is large enough to accommodate the working set of Linux, with the faster memory (DDR) and larger cache line size (32 bytes):
  - The performance of the processor improves by about 20%
  - The power consumed decreases by about 20%

Experiment 4: Typical Experimentation to Measure Inter-Platform Communication Latency and Bandwidth

- Communication Infrastructure bandwidth
  - Vary bus clock frequencies
    + Higher ➔ greater bandwidth – assuming intrinsic latencies are not violated
    + Communications Infrastructure width
- Latency due to
  - Arbitration of concurrent communication from various bus controllers on various interconnects
  - Bus transmission characteristics
- Meeting real-time schedules
  - Prioritization of messages
  - Decreasing latency
  - Increasing bandwidth

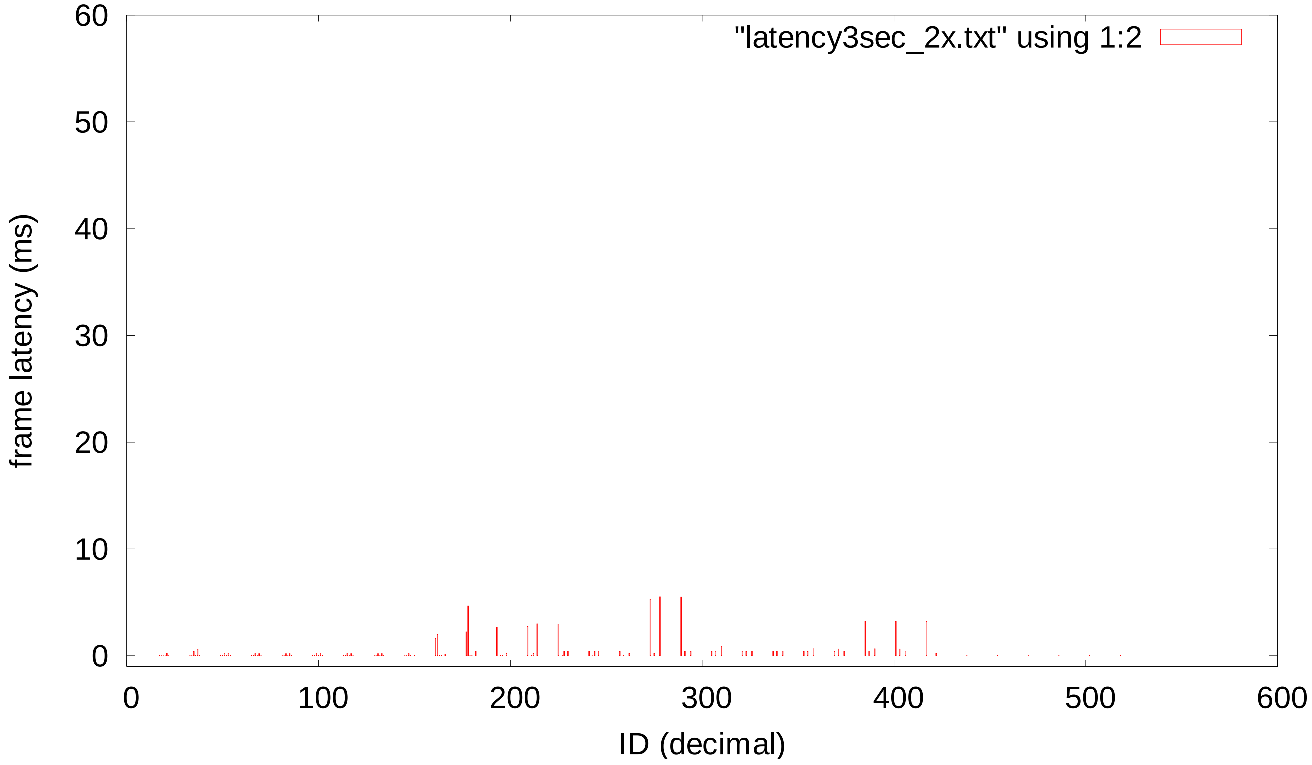
Experiment 4 Results: SubSystem Interconnect Bus Latencies for CAN Message ID @ 300kbits/sec



Some IDs have latencies that are higher than the periodic data transmission requirement ➔ Network has insufficient capabilities

Latency = (Start time of frame ID when it wins bus arbitration) – (Start time of frame ID for its first attempt to win bus arbitration)

Experiment 4-A *Doubling Bandwidth* Results: SubSystem Interconnect Bus Latencies for CAN Message ID @ 600kbits/sec



No IDs have latencies that are higher than the periodic data transmission requirement ➔ Network has sufficient capabilities

Doubling the bandwidth more than decreased the latencies by ½

The Next Steps
A Higher View of the System

An Opportunity to Structure Conceptual Abstractions to Understand Systems and Optimize Them

The Strategy
Structural Equation Modeling

- **Characterize the VSP in terms of concepts:**
  Develop concepts (such as, power, speed, noise, response latencies, cost, system function and constraints, …) and their comprehensive causal relationships with factors (and variables) and other concepts.

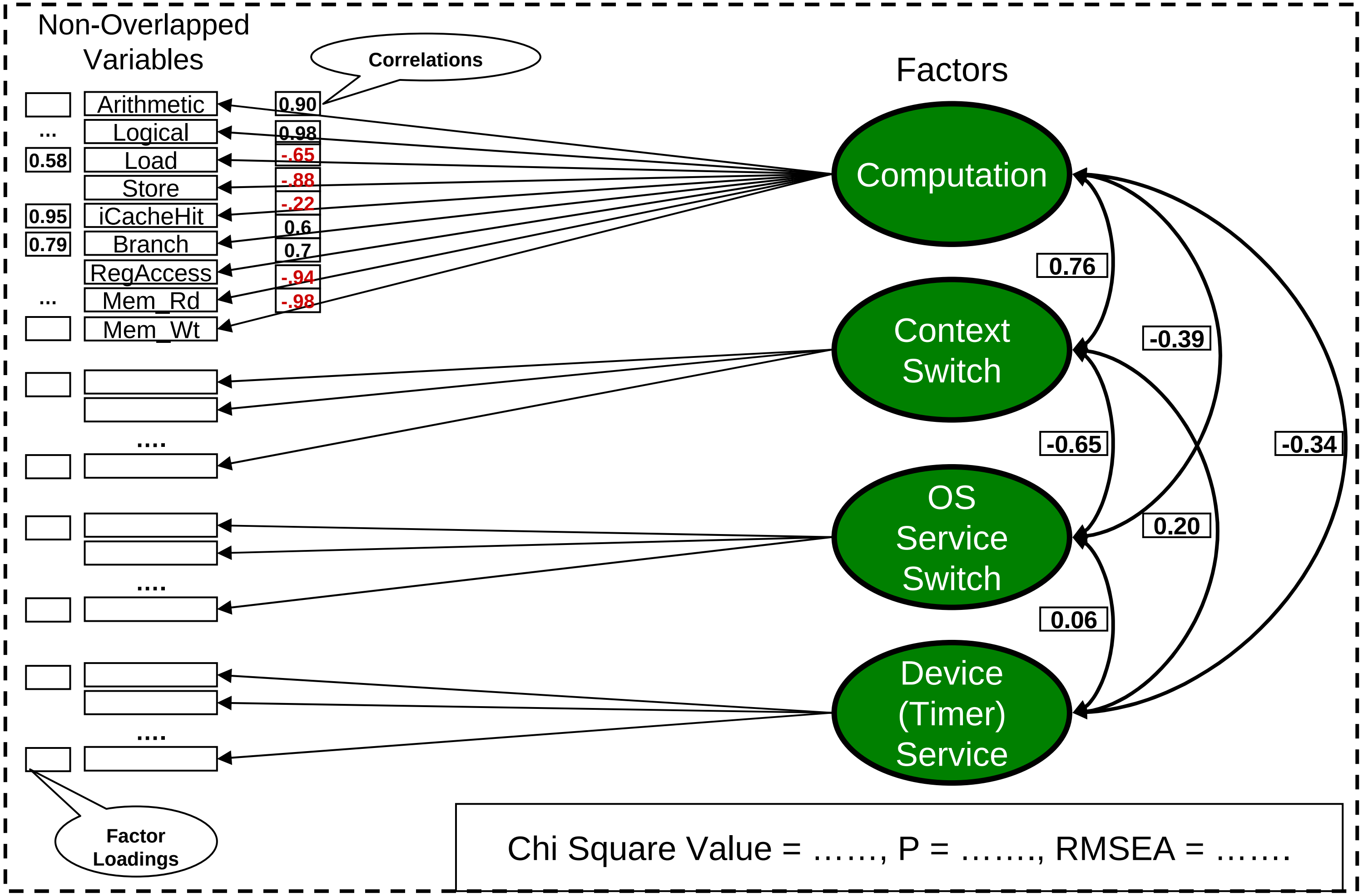
- **Specify the measurement model for each Concept:**
  Return to the factor analysis and specify for each factor one indicator variable (presumable the most highly correlated variable) together with a reliability estimate.

- **Formalize Overall concept as an equation in a set of equations – Structural Equation Model (SEM):**
  For each concept, translate the causal relation into a linear equation constituted from the factors, variables and other concepts In the relation.

- **Estimate the goodness of fit of the model with the original data:**
  The measurement model is used with the SEM model and the covariance matrix of the original variables to compute the goodness of fit of the SEM model to the data.

Measurement Modeling

Concept
Ex. Abstract Processor Efficiency Model

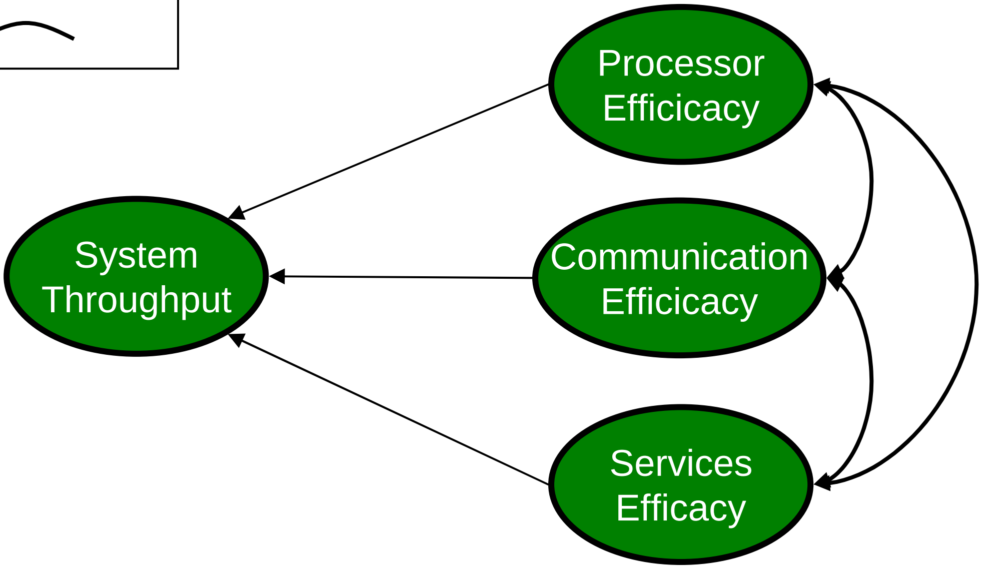


Chi Square Value = ……, P = ……., RMSEA = …….

Structural Equation Modeling

Best Fit Platform for Software Workload

Concept Relationships
Causal →
Correlation ⌒



Chi Square Value = ……, P = ……., RMSEA = …….

The Present & the Near Future

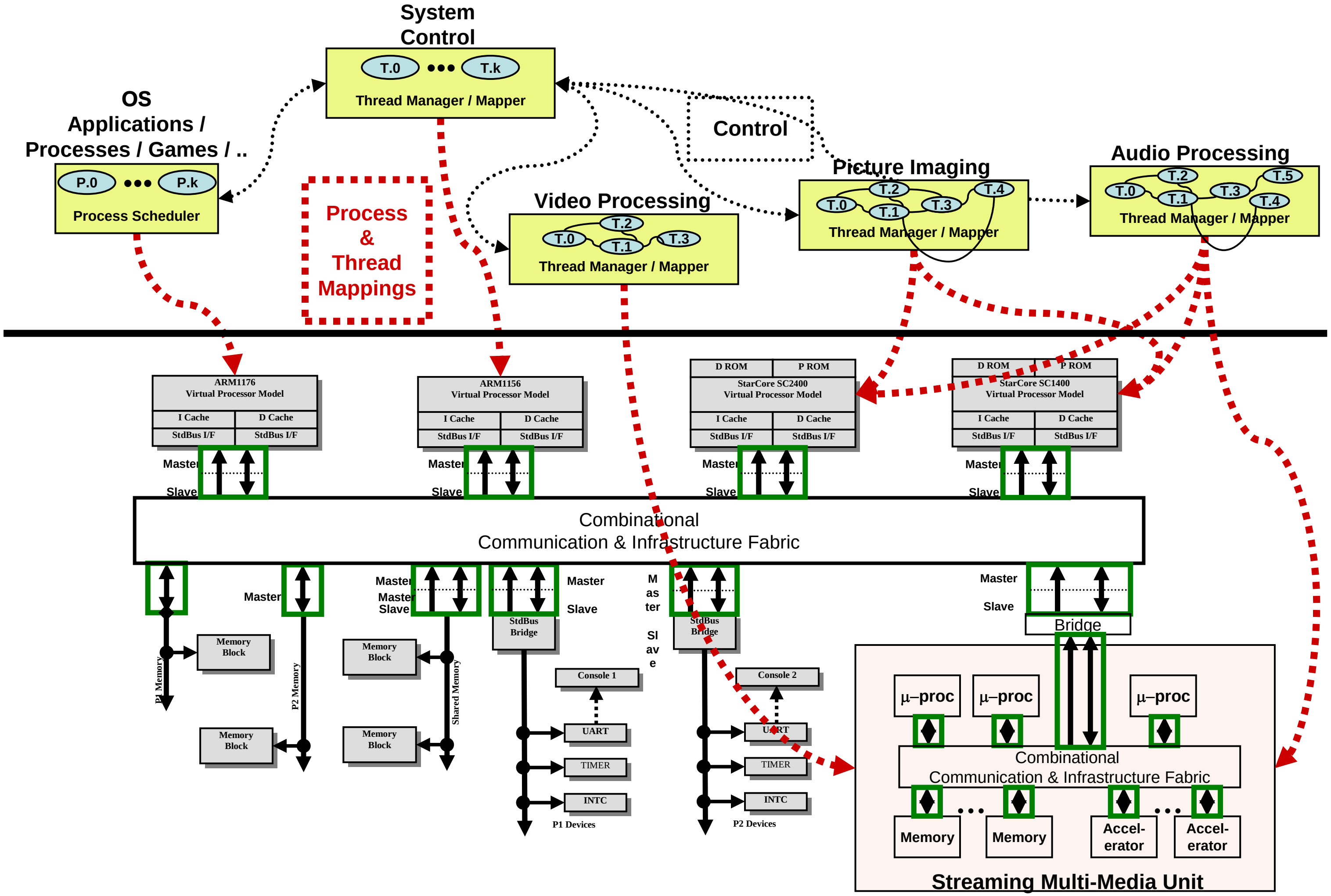
- The foundations of empirical systems analysis has been laid.

- We are in the process of building on the empirical work:
  - Formulating and verifying abstract Structural Equation Models to characterize performance and power
  - Correlating these abstract models with a simulating platform
  - Using the SEM and analysis to drive towards the optimization of systems, in which:
    - + We can demonstrate the balance between computation and communication empirically
  - Investigating statistical techniques used in *machine learning* to add further power to our quantitative analysis

Pictorially

This is where we believe we are headed …

QArk Current Objective:
Mapping Algorithms to a Typical Mobile System Architecture Dictated by Balanced Computation, Communications & Power



Thank You for Your Attention

& now ….

?’s