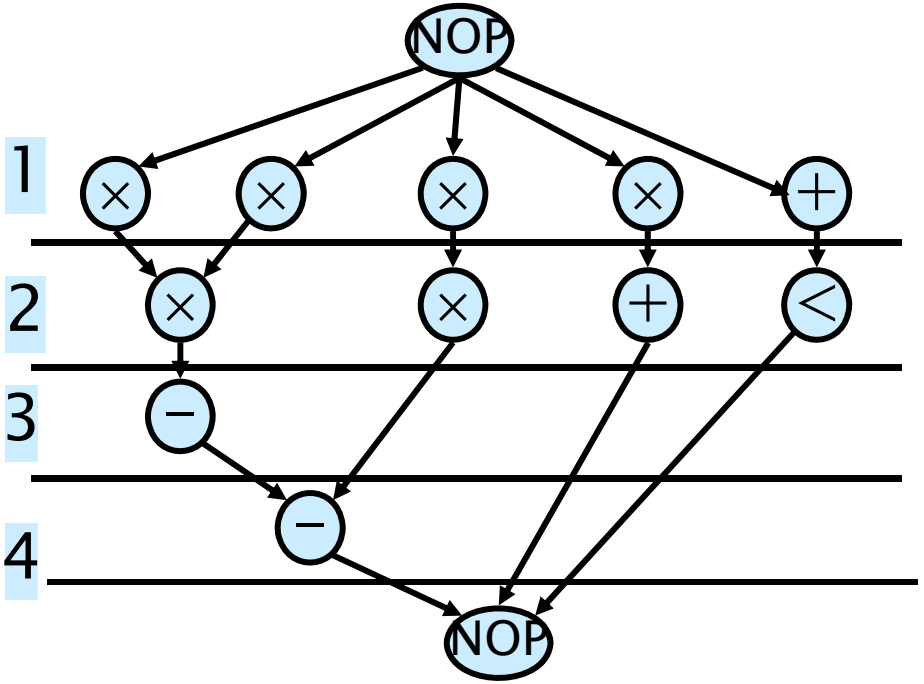
# Embedded Systems

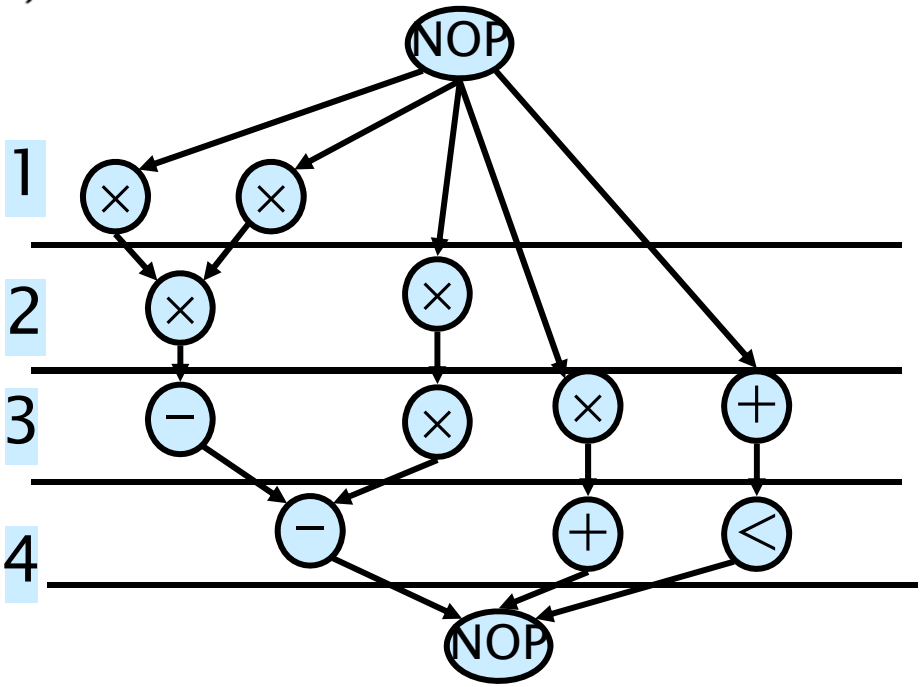
# ASAP Schedules

ASAP($G_S(V_S, E_S), w$) {
  $\tau(v_0) = 1$;
  REPEAT {
    Determine $v_i$ whose predec. are planed;
    $\tau(v_i) = \max\{\tau(v_j) + w(v_j)\ \forall (v_j, v_i) \in E_S\}$
  } UNTIL ($v_n$ is planned);
  RETURN ($\tau$);
}



# ALAP Schedules

ALAP$(G_S(V_S, E_S), w, L_{max})$ {
  $\tau(v_n) = L_{max} + 1$;
  REPEAT {
    Determine $v_i$ whose succ. are planed;
    $\tau(v_i) = \min\{\tau(v_j)\ \forall(v_i, v_j) \in E_S\} - w(v_i)$
  } UNTIL ($v_0$ is planned);
  RETURN ($\tau$);
}



# (Resource constrained) List Scheduling

**REVIEW**

Source: Teich: Dig. HW/SW Systeme

- List scheduling: extension of ALAP/ASAP method
- Preparation:
  - Greedy strategy (does NOT guarantee optimum solution)
  - Topological sort of task graph $G=(V,E)$
  - Computation of priority of each task:

    Possible priorities $u$:

    - Number of successors
    - Longest path
    - **Mobility** = $\tau$ (ALAP schedule)- $\tau$ (ASAP schedule)
      - Defined for each operation
      - Zero mobility implies that an operation can be started only at one given time step
      - Mobility greater than 0 measures span of time interval in which an operation may start → Slack on the start time

# Integer linear programming models

- Ingredients:
  - Cost function
  - Constraints

  } Involving linear expressions of integer variables from a set $X$

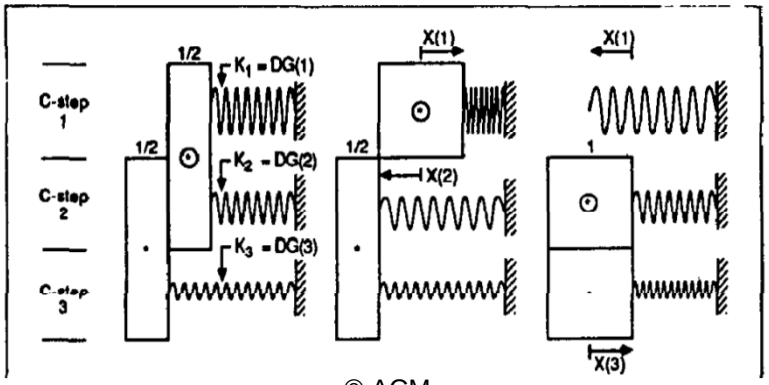
Cost function $$C = \sum_{x_i \in X} a_i x_i \text{ with } a_i \in R, x_i \in \mathbb{N} \quad (1)$$

Constraints: $$\forall j \in J : \sum_{x_i \in X} b_{i,j}\, x_i \geq c_j \text{ with } b_{i,j}, c_j \in \mathbb{R} \quad (2)$$

**Def**.: The problem of minimizing (1) subject to the constraints (2) is called an **integer linear programming (ILP) problem**.

If all $x_i$ are constrained to be either 0 or 1, the IP problem said to be a **0/1 integer linear programming problem**.

# (Time constrained) Force-directed scheduling



© ACM

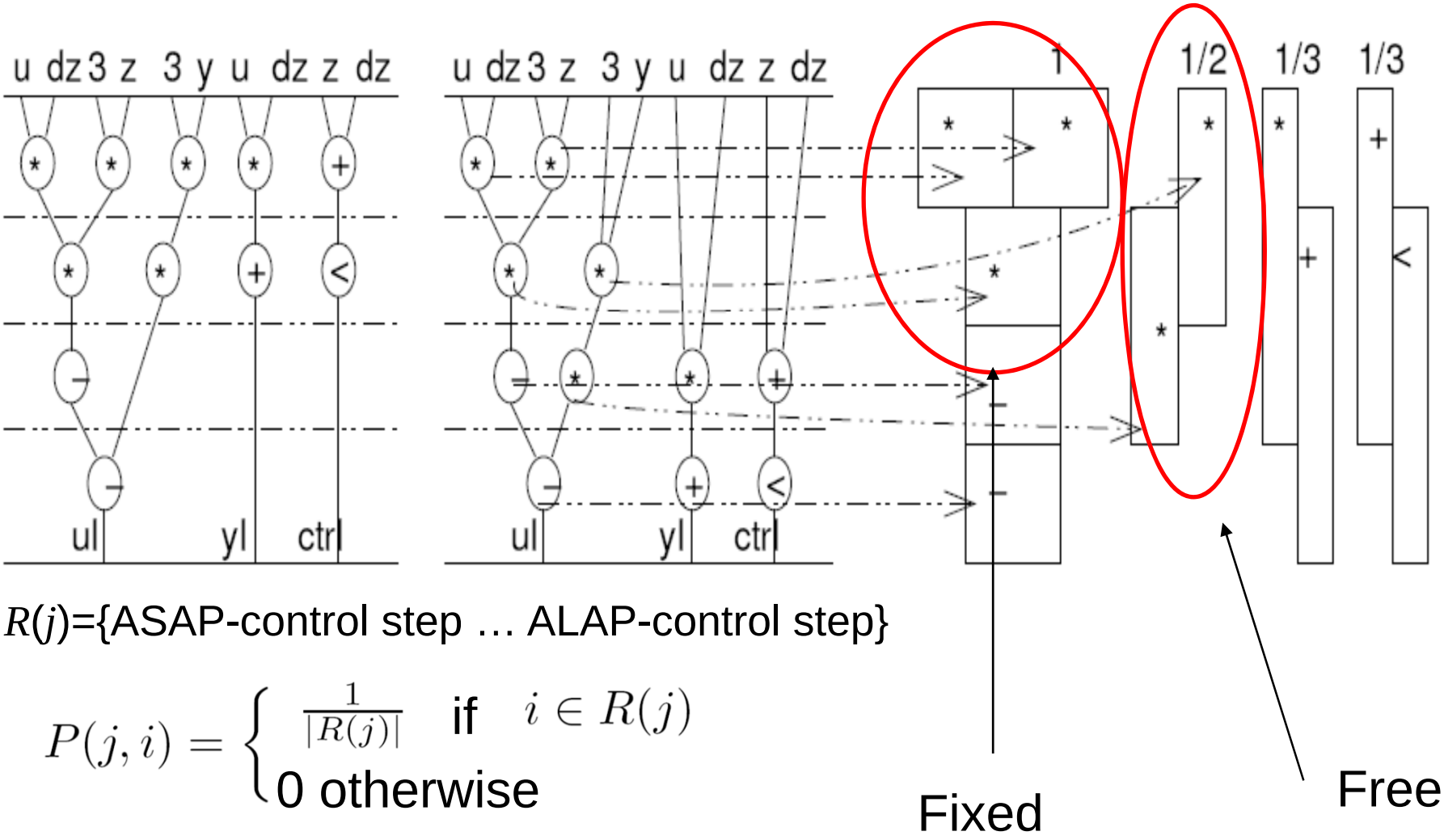
**REVIEW**

- Goal: balanced utilization of resources
- Based on spring model
- Originally proposed for high-level synthesis
- Force
  - Used as a priority function
  - Related to concurrency – sort operations for least force
  - Mechanical analogy: Force = constant x displacement
    - Constant = operation-type distribution
    - Displacement = change in probability

* [Pierre G. Paulin, J.P. Knight, Force-directed scheduling in automatic data path synthesis, *Design Automation Conference* (DAC), 1987, S. 195-202]

# REVIEW

## 1.Compute time frames $R(j)$
## 2. Compute "probability" $P(j,i)$ of assignment $j \rightarrow i$




$R(j)$={ASAP-control step … ALAP-control step}

$$P(j,i) = \begin{cases} \frac{1}{|R(j)|} & \text{if} \quad i \in R(j) \\ 0 & \text{otherwise} \end{cases}$$

- Architecture Synthesis
- HW/SW Codesign
- Power Aware Computing

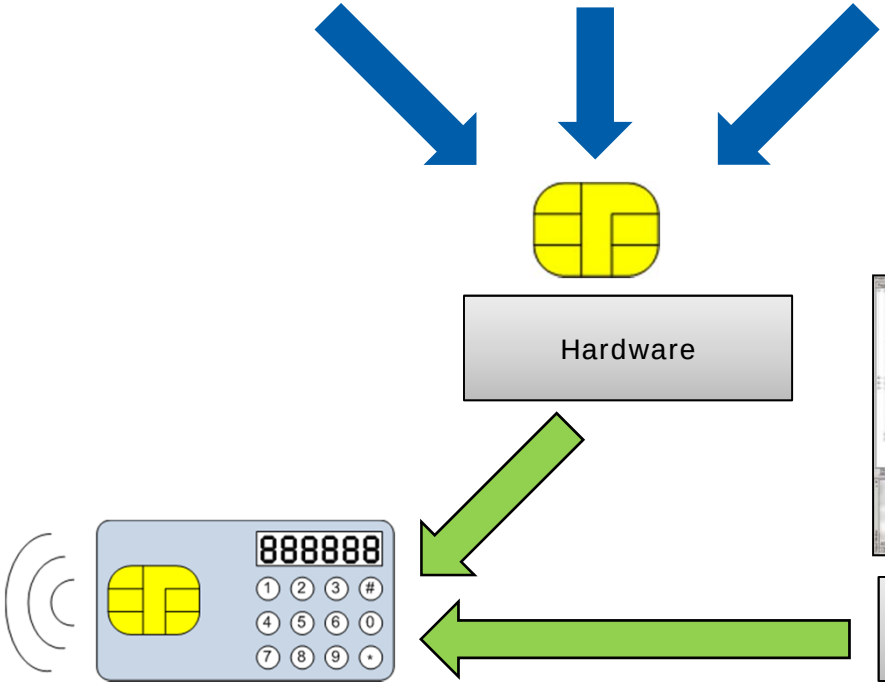
- 3.2.2011 Lecture by Bernd Finkbeiner, Head of Reactive Systems Group at Saarland University(http://react.cs.uni-sb.de/

# Codesign Definition and Key Concepts

- Codesign
  - The meeting of system-level objectives by exploiting the trade-offs between hardware and software in a system through their concurrent design
- Key concepts
  - Concurrent: hardware and software developed at the same time on parallel paths
  - Integrated: interaction between hardware and software development to produce design meeting performance criteria and functional specs

# Low Power HW/SW Co-Design of Smart Cards: Approach

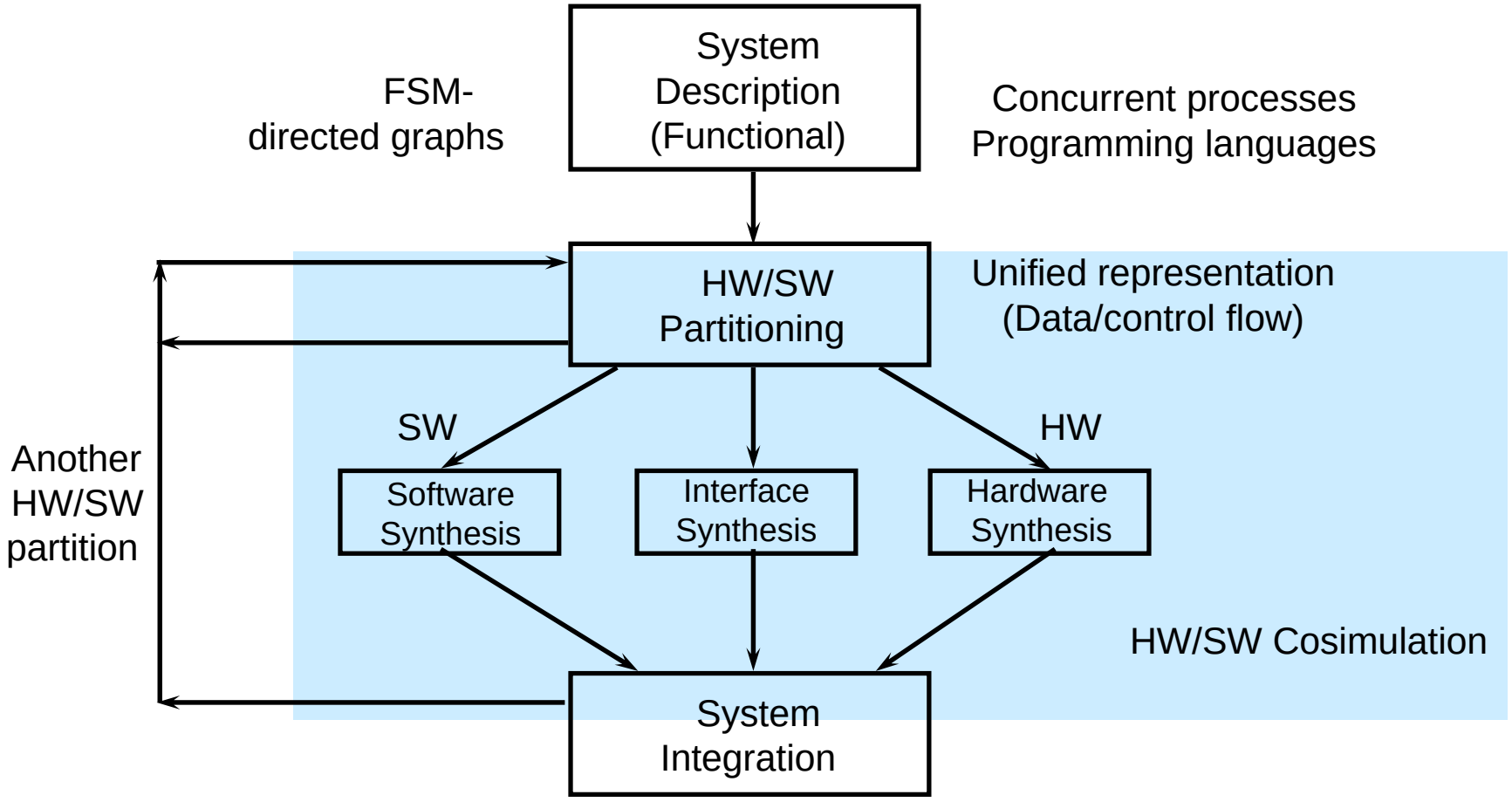
| Selection of Silicon Technology | Selection of Architecture | Apply Low Power Measures and Tools |
|---|---|---|
| Technology node (130nm, 90nm, ...) | CPU architecture (8/16/32-bit) | Low power design (supply voltage,...) |
| Non-volatile memories (Flash, EEPROM,...) | Security level | Automated tools (clock gating, ...) |



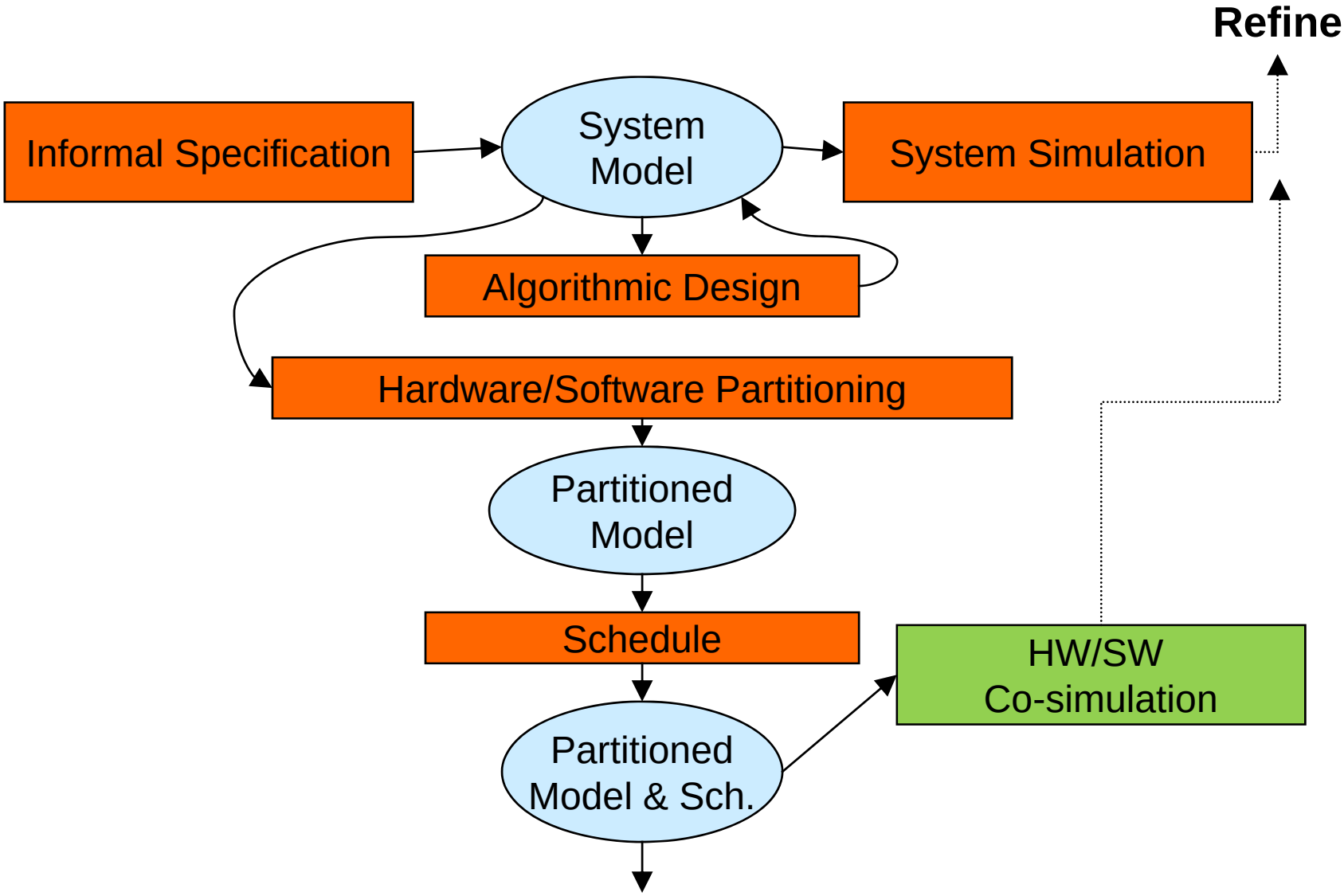

| Software Architecture |
|---|
| SW power management |
| High performance optimizations |

| Software Tools |
|---|
| Power emulation |
| Optimized compiler |

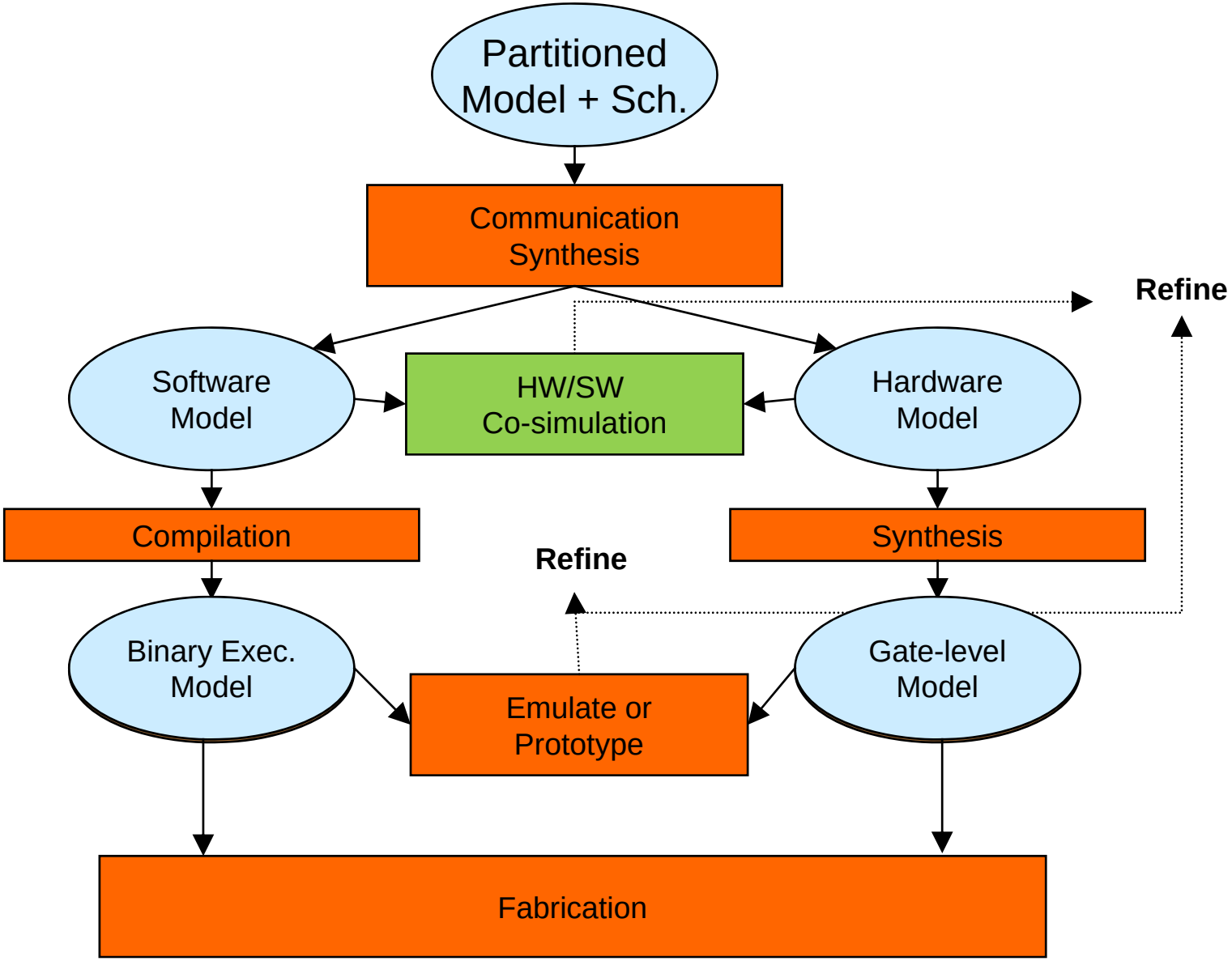
# Typical Codesign Process



# Co-design Flow in more detail



# Co-design Flow Cont…



# Categories of Codesign Problems

- Codesign of embedded systems
  - Usually consist of sensors, controller, and actuators
  - Are reactive systems
  - Usually have real-time constraints
  - Usually have dependability constraints
- Codesign of ISAs
  - Application-specific instruction set processors (ASIPs)
  - Compiler and hardware optimization and trade-offs
- Codesign of Reconfigurable Systems
  - Systems that can be personalized after manufacture for a specific application

# Main Tasks of the Codesign Problem

- Specification of the system
- Hardware/Software Partitioning
  - Architectural assumptions - type of processor, interface style between hardware and software, etc.
  - Partitioning objectives - maximize speedup, latency requirements, minimize size, cost, etc.
  - Partitioning strategies - high level partitioning by hand, automated partitioning using various techniques, etc.
- Scheduling
  - Operation scheduling in hardware
  - Instruction scheduling in compilers
  - Process scheduling in operating systems
- Modeling/Simulation of the hardware/software system during the design process

# Issues in Partitioning

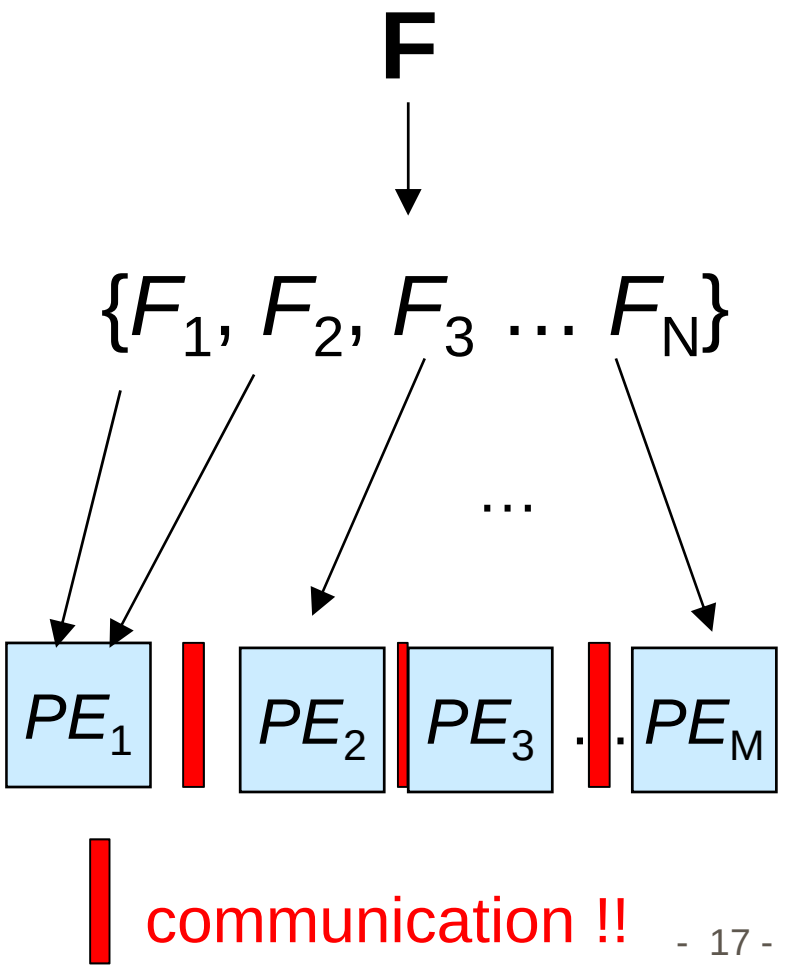
- Specification abstraction level
- Granularity
- System-component allocation
- Metrics and estimations
- Partitioning algorithms
- Objective and closeness functions
- Partitioning algorithms
- Flow of control and designer interaction

# Hardware Software Partitioning

- Decompose (i.e., partition) the function F of the system into N sub-functions $F_1, F_2, F_3 \ldots F_N$
- Decompose the constraints and design objectives of the system into sub-constraints and design sub-objectives
- Cluster $F_1, F_2, F_3 \ldots F_N$ into M partitions to run on M processors elements (mapping)
- Given:

  **F** = { $F_1, F_2, F_3 \ldots F_N$ } ;
  **P** = { $P_1, P_2, P_3 \ldots P_M$ }

- Find a lowest cost partition (cluster), as computed by an objective function
- Exhaustive approach $O(M^N)$



# Computation of Metrics

- Two approaches to computing metrics
  - Creating a detailed implementation
    - Produces accurate metric values
    - Impractical as it requires too much time
  - Creating a rough implementation
    - Includes the major register transfer components of a design
    - Skips details such as precise routing or optimized logic, which require much design time
    - Determining metric values from a rough implementation is called estimation

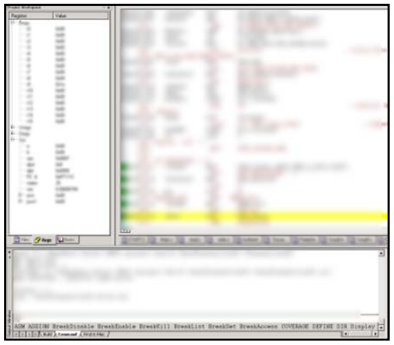
# Estimation

- Cost depends on components selected to implement the application!
  - Software Processors: PowerPC, ARM, Pentium, …
  - Hardware: FPGAs, ASIC blocks, …
  - Communication Infrastructure: buses, networks-on-chip, p2p links, …

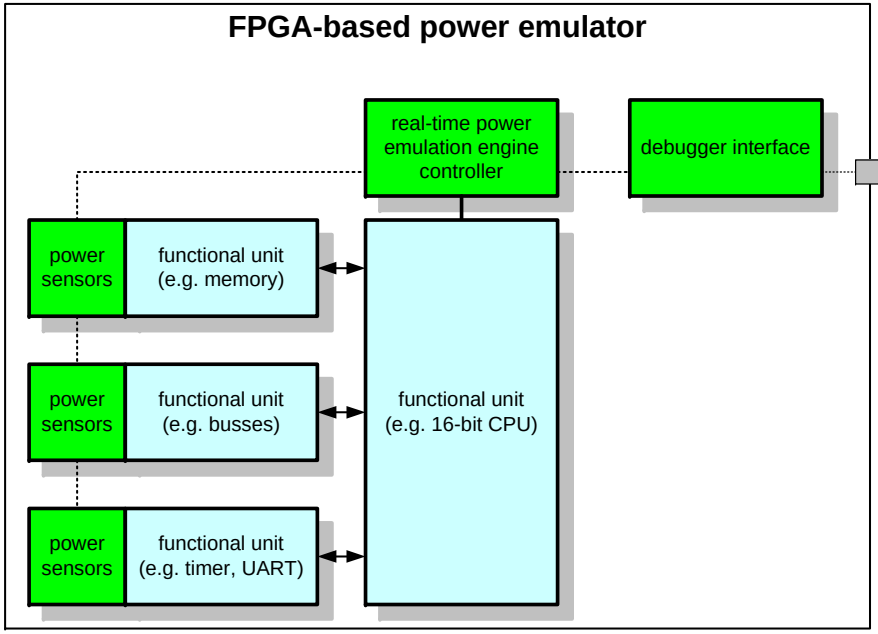
- Profiling tools are used prior to partitioning to determine cost and also to determine critical parts of application
  - obtain performance (or power, area, …) metrics of the system
  - helps the designer optimize the design and decide whether to implement certain functions in hardware or software

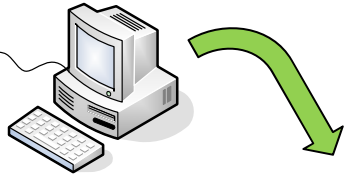
# Poweremulation

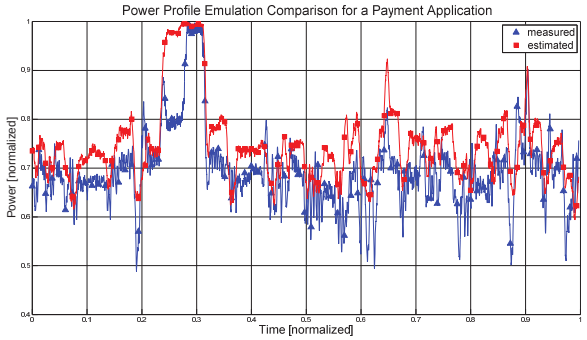
- POWERHOUSE vision



- Implementation of power model on emulation platform: ***Power emulation* (PE)**
- Generate power estimates as a by-product of functional emulation during system run-time
- Visualize and evaluate data within a software IDE
- Improve power-awareness based on power feedback





# Objective and Closeness Functions

- Multiple metrics, such as cost, power, and performance are weighed against one another
  - An expression combining multiple metric values into a single value that defines the quality of a partition is called an Objective Function
  - The value returned by such a function is called cost
  - Because many metrics may be of varying importance, a weighted sum objective function is used (and constr.)
    - e.g Cost = c1 * F(area, area_constr)
      + c2 * F(delay, delay_constr)
      + c3 * F(power, power_constr)

# Partitioning Algorithm Classes

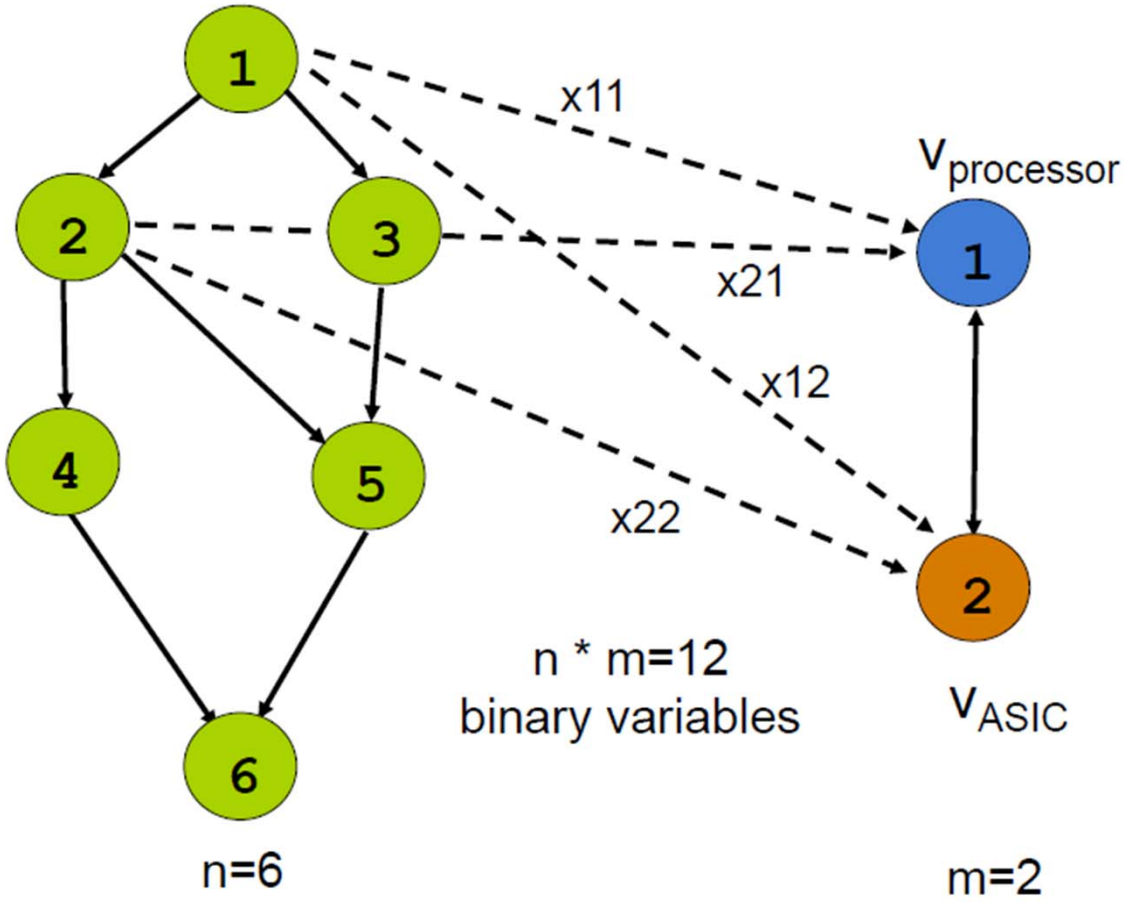
- Constructive algorithms
  - Group objects into a complete partition
  - Use closeness metrics to group objects, hoping for a good partition
- Iterative algorithms
  - Modify a complete partition in the hope that such modifications will improve the partition
  - Use an objective function to evaluate each partition
  - Yield more accurate evaluations than closeness functions used by constructive algorithms
- In practice, a combination of constructive and iterative algorithms is often employed

# Partitioning Methods

- Exact methods
  - Integer Linear Programming (ILP)
  - …
- Heuristic methods
  - Constructive methods
    - Random mapping
    - Hierarchical clustering
  - Iterative methods
    - Kernighan-Lin Algorithm
    - Simulated Annealing
    - …

# ILP HW/SW Partitioning

Example from Christian Plessl, Universität Paderborn




| task | SW cost | HW cost |
|---|---|---|
| 1 | 80 | 320 |
| 2 | 240 | 170 |
| 3 | 710 | 120 |
| 4 | 130 | 20 |
| 5 | 100 | 400 |
| 6 | 80 | 260 |

# ILP HW/SW Partitioning

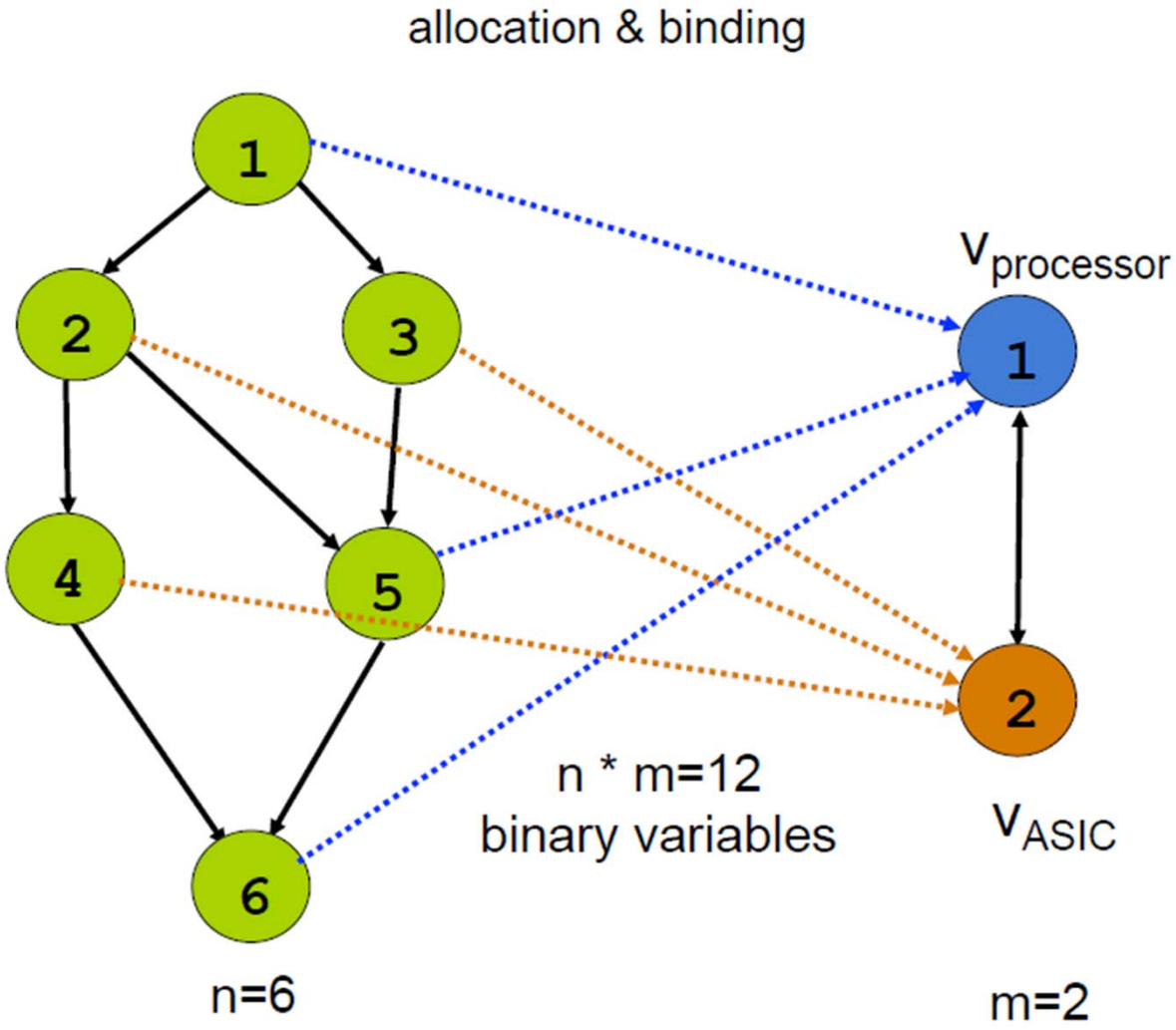
cost table
(all bindings possible)

| task | SW cost | HW cost |
|---|---|---|
| 1 | 80 | 320 |
| 2 | 240 | 170 |
| 3 | 710 | 120 |
| 4 | 130 | 20 |
| 5 | 100 | 400 |
| 6 | 80 | 260 |

```
/************************/
/* objective function   */
/************************/

/*   c1s = 80   c1h = 320 */
/*   c2s = 240  c2h = 170 */
/*   c3s = 710  c3h = 120 */
/*   c4s = 130  c4h = 20  */
/*   c5s = 100  c5h = 400 */
/*   c6s = 80   c6h = 260 */

min: 80 x11 + 320 x12 + 240 x21 + 170 x22 + 710 x31  +
120 x32 +  130 x41 + 20 x42 + 100 x51 + 400 x52 + 80 x61
+ 260 x62;


/*****************************/
/* unique mapping constraints */
/*****************************/
x11 + x12 = 1;
x21 + x22 = 1;
x31 + x32 = 1;
x41 + x42 = 1;
x51 + x52 = 1;
x61 + x62 = 1;
```

```
/***********************/
/* variables in {0,1} */
/***********************/
x11 <= 1;
x12 <= 1;
x21 <= 1;
x22 <= 1;
x31 <= 1;
x32 <= 1;
x41 <= 1;
x42 <= 1;
x51 <= 1;
x52 <= 1;
x61 <= 1;
x62 <= 1;



/*********************/
/* integer variables */
/*********************/
int x11;
int x12;
int x21;
int x22;
int x31;
int x32;
int x41;
int x42;
int x51;
int x52;
int x61;
int x62;
```

# ILP HW/SW Partitioning




cost table

| task | SW cost | HW cost |
|---|---|---|
| 1 | 80 | 320 |
| 2 | 240 | 170 |
| 3 | 710 | 120 |
| 4 | 130 | 20 |
| 5 | 100 | 400 |
| 6 | 80 | 260 |

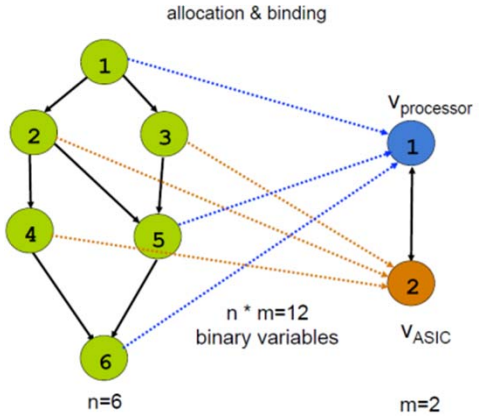
total cost = 570

# ILP HW/SW Partitioning

- Constraint on the hardware cost
  - cost of all tasks mapped to hardware must not exceed 300

$$\sum_{i=1}^{6} c_{i,2} \cdot x_{i,2} \leq 300$$

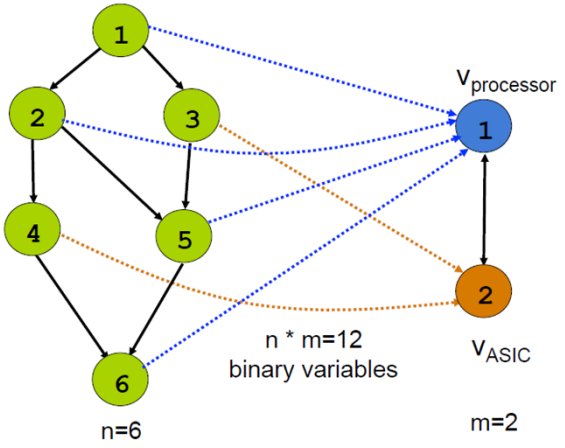
```
/*****************************/
/* hardware cost constraint */
/*****************************/
320 x12 + 170 x22 + 120 x32 + 20 x42 + 400 x52 + 260 x62 <= 300;
```




cost table

| task | SW cost | HW cost |
|---|---|---|
| 1 | 80 | 320 |
| 2 | 240 | 170 |
| 3 | 710 | 120 |
| 4 | 130 | 20 |
| 5 | 100 | 400 |
| 6 | 80 | 260 |

total cost = 570




| task | SW cost | HW cost |
|---|---|---|
| 1 | 80 | 320 |
| 2 | 240 | 170 |
| 3 | 710 | 120 |
| 4 | 130 | 20 |
| 5 | 100 | 400 |
| 6 | 80 | 260 |

constraint: HW cost <= 300

total cost = 640

# Partitioning Methods

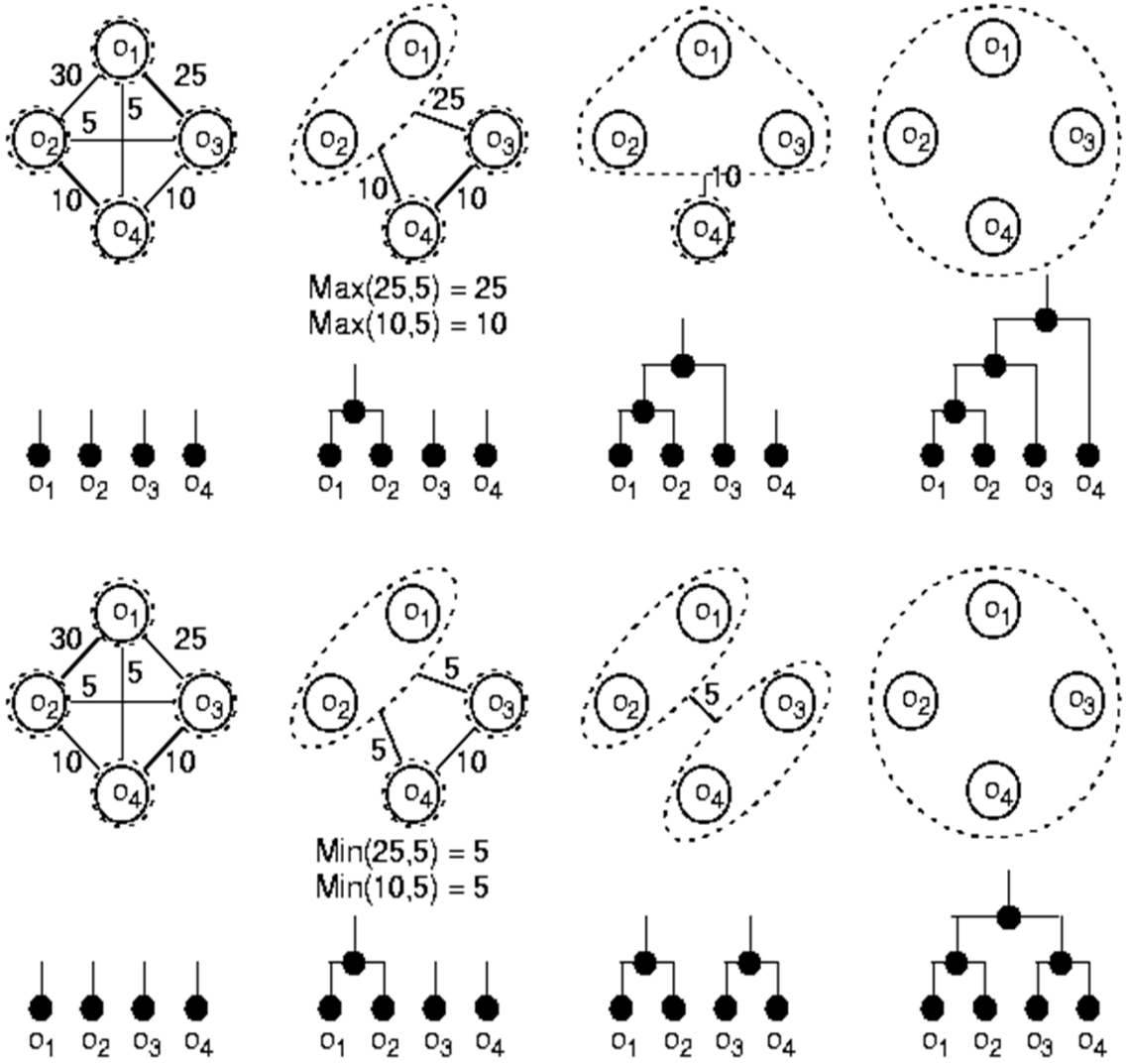
- Exact methods
  - Enumeration
  - Integer Linear Programming (ILP)
- Heuristic methods
  - Constructive methods
    - Random mapping
    - Hierarchical clustering
  - Iterative methods
    - Kernighan-Lin Algorithm
    - Simulated Annealing
    - …

# Constructive Methods

- **Random mapping**
  - Each object randomly assigned to some block
  - Used to find starting partition for iterative methods

- **Hierarchical clustering**
  - Assumes closeness function: determines how desirable it is to group two objects
  - Start with singleton blocks
  - Repeat until termination criterion (e.g., desired number of blocks reached)
    - Compute closeness of blocks (average closeness of object pairs)
    - Find pair of closest blocks
    - Merge blocks
  - Difficulty: find proper closeness function

# Example: Hierarchical Clustering

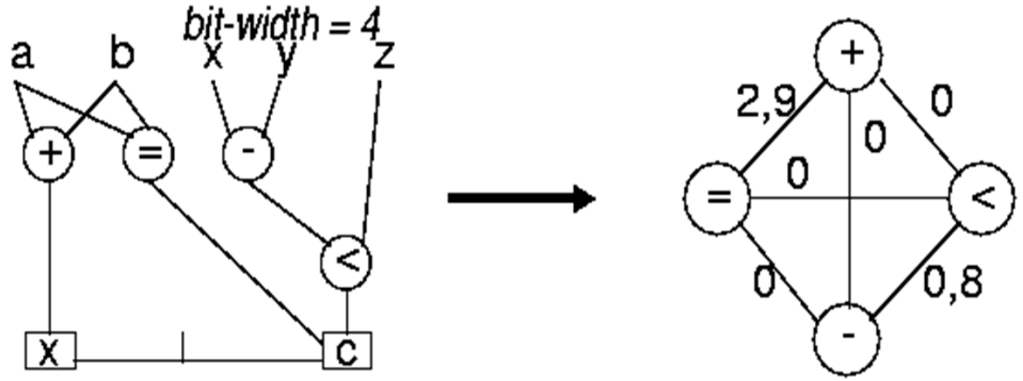


# Case Study: YSC (IBM)

- **Yorktown Silicon Compiler:** functional partitioning of hardware
- **Input:** functional description on the level of arithmetic and logical expressions
- **Target:** partitioning to several chips
- **Abstraction level:** functional units of datapaths (ALUs, registers)
- **Method:** hierarchical clustering

$$closeness(p_i, p_j) = \left( \frac{sharedwires(p_i, p_j)}{maxwires} \right)^{c_1} \cdot \left( \frac{maxsize}{\min\{size(p_i), size(p_j)\}} \right)^{c_2} \cdot \left( \frac{maxsize}{size(p_i) + size(p_j)} \right)$$

# Closeness function




| # Transistors | |
|---|---|
| + | 120 |
| = | 140 |
| - | 160 |
| < | 180 |

$$\text{Closeness}(+,=) = \frac{8+0}{8} \times \frac{300}{120} \times \frac{300}{120+140} = 2,9$$

$$\text{Closeness}(-,<) = \frac{0+4}{8} \times \frac{300}{160} \times \frac{300}{160+180} = 0,8$$

$$closeness(p_i, p_j) = \left(\frac{sharedwires(p_i, p_j)}{maxwires}\right)^{c_1} \cdot \left(\frac{maxsize}{\min\{size(p_i), size(p_j)\}}\right)^{c_2} \cdot \left(\frac{maxsize}{size(p_i) + size(p_j)}\right)$$

# Partitioning Methods

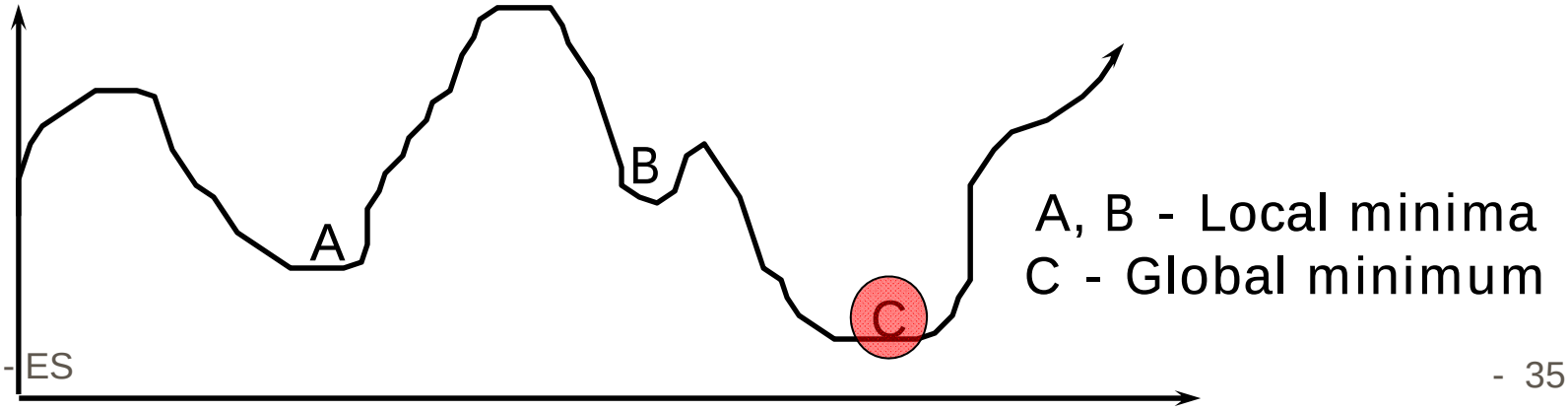
- Exact methods
  - Enumeration
  - Integer Linear Programming (ILP)
- Heuristic methods
  - Constructive methods
    - Random mapping
    - Hierarchical clustering
  - Iterative methods
    - Greedy
    - Kernighan-Lin Algorithm
    - Simulated Annealing
    - …

# Iterative Partitioning Algorithms

- Two broad categories:
  - Greedy algorithms
    - Only accept moves that decrease cost
    - Can get trapped in local minima
  - Hill-climbing algorithms
    - Allow moves in directions increasing cost (retracing)
      - Through use of stochastic functions
    - Can escape local minima
    - E.g., simulated annealing

# Iterative Partitioning Algorithms

- The computation time in an iterative algorithm is spent evaluating large numbers of partitions
- Iterative algorithms differ from one another primarily in the ways in which they modify the partition and in which they accept or reject bad modifications
- The goal is to find global minimum while performing as little computation as possible



# HW/SW Partitioning

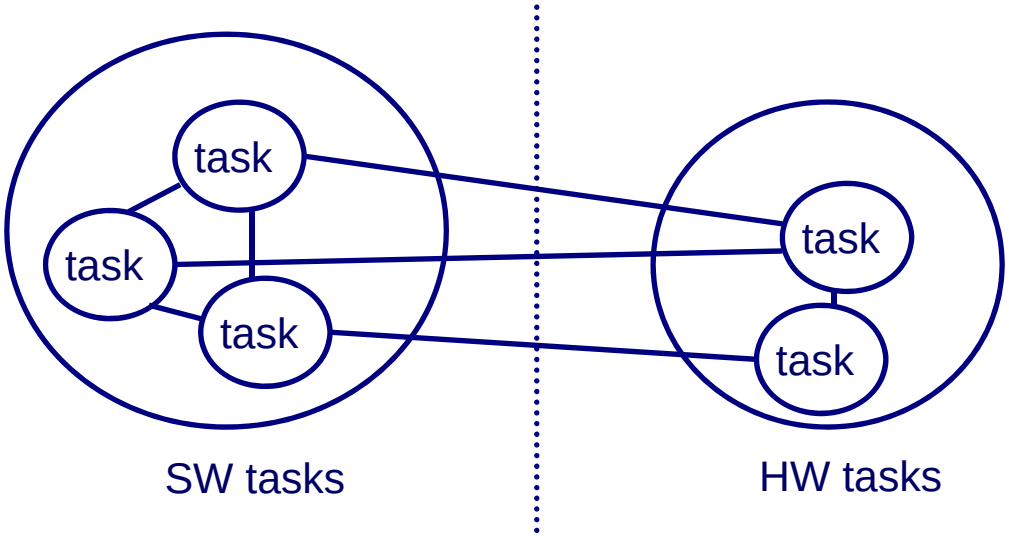
- Special case: Bi-partitioning $P=\{p_{SW}, p_{HW}\}$
- Software-oriented approach: $P=\{O,\varnothing\}$
  - In software, all functions can be realized
  - Performance might be too low $\Rightarrow$ migrate objects to HW
- Hardware-oriented approach: $P=\{\varnothing,O\}$
  - In hardware, performance is OK
  - Cost might be too high $\Rightarrow$ migrate objects to SW

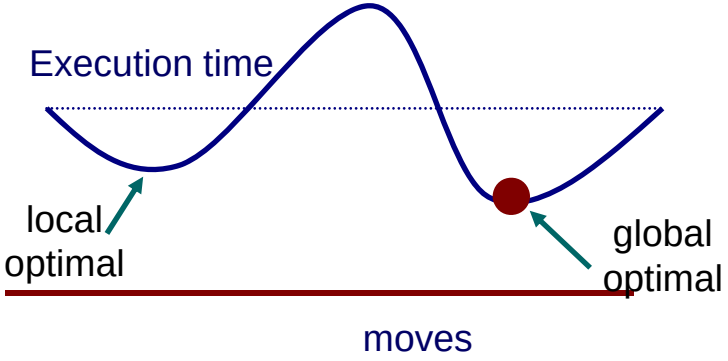
# Greedy Hw/Sw Partitioning

Migration of objects to the other block (HW/SW) until no more improvement

```
repeat
  begin
  P'=P;
  for i=1 to n
      begin
        if (cost(move(P,o_i) < cost(P))
        then P':=move(P,o_i);
      end;
  end;
until (P==P')
```

# Kernighan-Lin (Min-Cut)

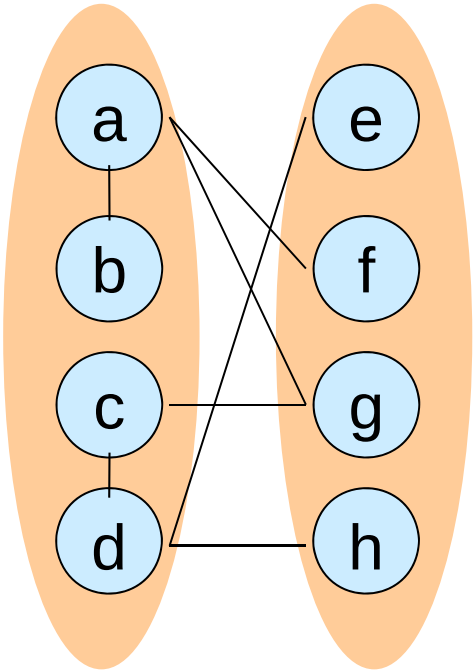


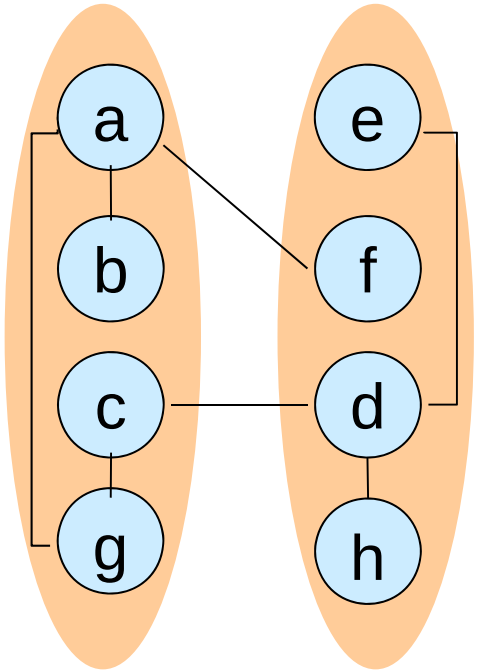



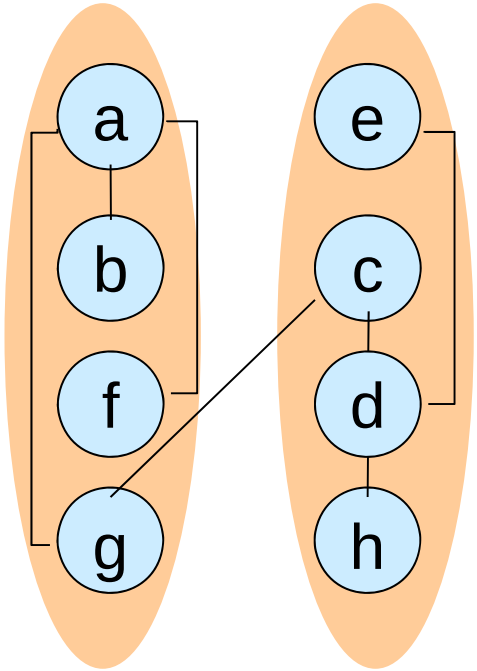

**Kernighan/Lin – Fidducia/Mattheyses algorithm**

- Start with all task vertices free to swap/move (*unlocked*)
- Label each possible swap/move with immediate change in execution time that it causes (*gain*)
- Iteratively select and execute a swap/move with highest gain (**whether positive or negative**); *lock* the moving vertex (i.e., cannot move again during the pass),
- Best solution seen during the pass is adopted as starting solution for next pass

# Example



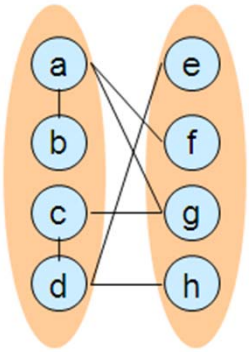




| Step # | Object pair | Cost reduction | Cut cost |
|---|---|---|---|
| 0 | | | 5 |
| 1 | {d,g} | 3 | 2 |
| 2 | {c,f} | 1 | 1 |
| 3 | {b,h} | -2 | 3 |
| 4 | {a,e} | -2 | 5 |

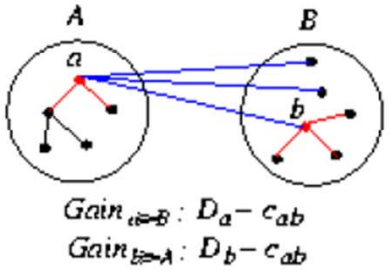
Questions: How to compute cost reduction? What pairs to be swapped?

Consider the change of internal & external connections.

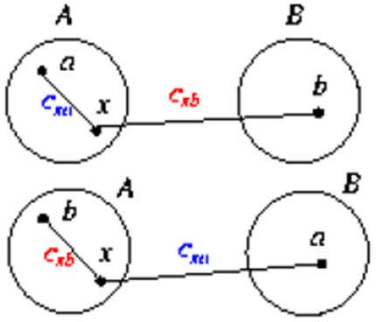
# Computing the cost reduction




- External cost of a$\in A$: $E_a=\sum_{v\in B} c_{av}$
- Internal cost of $a\in A$: $I_a=\sum_{v\in A} c_{av}$
- Cost reduction for moving $a$ : $D_a=E_a-I_a$
- Cost reduction for swapping $a$ and $b$: $g_{ab}=D_a+D_b-2c_{ab}$
- Update to $D$-values when a and b are swapped:
  $D'_x = D_x + 2c_{xa} - 2c_{xb}$ for all $x\in A-\{a\}$
  $D'_y = D_y + 2c_{yb} - 2c_{ya}$ for all $y\in B-\{b\}$




Internal cost vs. External cost




| before swap | after swap | $\Delta C$ |
|---|---|---|
| $-c_{xa}$ | $+c_{xa}$ | $+2c_{xa}$ |
| $+c_{xb}$ | $-c_{xb}$ | $-2c_{xb}$ |

updating D−values

# Kernighan-Lin

- **Repeat**
  - Compute $D_v$ für all objects
  - Mark all vertices as unlocked
  - **For** $i$=1 to $n/2$ **do**
    - Compute $g_{ab}$ for all pairs $a,b$
    - Pick unlocked $a_i,b_i$ with largest $g_{ab,i}$
    - Mark $a_i,b_i$ as locked
    - Store gain
    - Update $D_v$ für all objects
  - Find $k$ such that $G_k=\sum_{i=1}^{k} g_{ab,i}$ is maximal
  - **If** $G_k>0$, **then** move $a_1,\ldots,a_k$ from A to B and $b_1,\ldots,b_k$ from B to A.
- **Until** $G_k \leq 0$

$O(n2)$

$O(n^3)$.

Suppose the repeat loop terminates after $r$ passes.

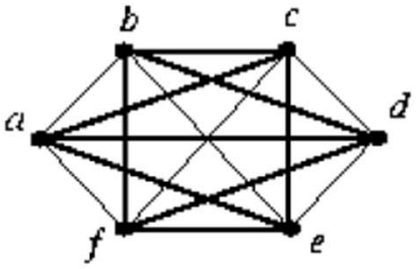
The total running time: $O(rn^3)$
Polynomial-time algorithm?

# Weighted Example

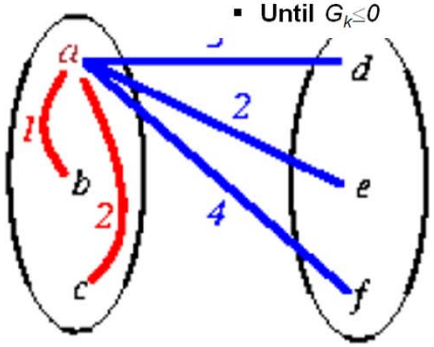
- **Repeat**
  - Compute $D_v$ für all objects
  - Mark all vertices as unlocked
  - **For** $i$=1 to $n/2$ **do**
    - Compute $g_{ab}$ for all pairs $a,b$
    - Pick unlocked $a_i,b_i$ with largest $g_{ab,i}$
    - Mark $a_i,b_i$ as locked
    - Store gain
    - Update $D_v$ für all objects
  - Find $k$ such that $G_k=\sum^k_{i=1} g_{ab,i}$ is maximal
  - **If** $G_k>0$, **then** move $a_1,\ldots,a_k$ from A to B and $b_1,\ldots,b_k$ from B to A.
- **Until** $G_k \leq 0$

A={a,b,c}

B={d,e,f}




| | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| a | 0 | 1 | 2 | 3 | 2 | 4 |
| b | 1 | 0 | 1 | 4 | 2 | 1 |
| c | 2 | 1 | 0 | 3 | 2 | 1 |
| d | 3 | 4 | 3 | 0 | 4 | 3 |
| e | 2 | 2 | 2 | 4 | 0 | 2 |
| f | 4 | 1 | 1 | 3 | 2 | 0 |




costs associated with $a$

*Initial cut cost* = $(3+2+4)+(4+2+1)+(3+2+1) = 22$

- Iteration 1:

$$I_a = 1+2 = 3; \quad E_a = 3+2+4 = 9; \quad D_a = E_a - I_a = 9-3 = 6$$
$$I_b = 1+1 = 2; \quad E_b = 4+2+1 = 7; \quad D_b = E_b - I_b = 7-2 = 5$$
$$I_c = 2+1 = 3; \quad E_c = 3+2+1 = 6; \quad D_c = E_c - I_c = 6-3 = 3$$
$$I_d = 4+3 = 7; \quad E_d = 3+4+3 = 10; \quad D_d = E_d - I_d = 10-7 = 3$$
$$I_e = 4+2 = 6; \quad E_e = 2+2+2 = 6; \quad D_e = E_e - I_e = 6-6 = 0$$
$$I_f = 3+2 = 5; \quad E_f = 4+1+1 = 6; \quad D_f = E_f - I_f = 6-5 = 1$$

# g-Value Computation

- **Repeat**
  - Compute $D_v$ für all objects
  - Mark all vertices as unlocked
  - **For** $i$=1 to $n/2$ **do**
    - Compute $g_{ab}$ for all pairs $a,b$
    - Pick unlocked $a_i,b_i$ with largest $g_{ab,i}$
    - Mark $a_i,b_i$ as locked
    - Store gain
    - Update $D_v$ für all objects
  - Find $k$ such that $G_k=\sum_{i=1}^{k} g_{ab,i}$ is maximal
  - **If** $G_k>0$, **then** move $a_1,\ldots,a_k$ from A to B and $b_1,\ldots,b_k$ from B to A.
- **Until** $G_k \leq 0$

- Iteration 1:

$$
\begin{array}{lll}
I_a = 1+2 = 3; & E_a = 3+2+4 = 9; & D_a = E_a - I_a = 9-3 = 6 \\
I_b = 1+1 = 2; & E_b = 4+2+1 = 7; & D_b = E_b - I_b = 7-2 = 5 \\
I_c = 2+1 = 3; & E_c = 3+2+1 = 6; & D_c = E_c - I_c = 6-3 = 3 \\
I_d = 4+3 = 7; & E_d = 3+4+3 = 10; & D_d = E_d - I_d = 10-7 = 3 \\
I_e = 4+2 = 6; & E_e = 2+2+2 = 6; & D_e = E_e - I_e = 6-6 = 0 \\
I_f = 3+2 = 5; & E_f = 4+1+1 = 6; & D_f = E_f - I_f = 6-5 = 1
\end{array}
$$

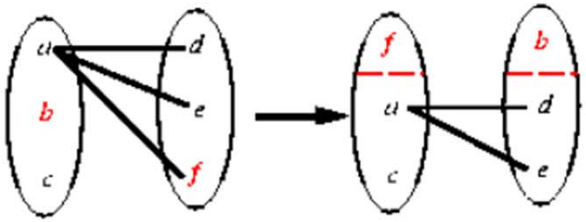
- $g_{xy} = D_x + D_y - 2c_{xy}$.

$$
\begin{aligned}
g_{ad} &= D_a + D_d - 2c_{ad} = 6+3-2\times 3 = 3 \\
g_{ae} &= 6+0-2\times 2 = 2 \\
g_{af} &= 6+1-2\times 4 = -1 \\
g_{bd} &= 5+3-2\times 4 = 0 \\
g_{be} &= 5+0-2\times 2 = 1 \\
g_{bf} &= 5+1-2\times 1 = 4 \ (maximum) \\
g_{cd} &= 3+3-2\times 3 = 0 \\
g_{ce} &= 3+0-2\times 2 = -1 \\
g_{cf} &= 3+1-2\times 1 = 2
\end{aligned}
$$

- Swap $b$ and $f$! $(\hat{g_1} = 4)$

# D-Value Computation

- **Repeat**
  - Compute $D_v$ für all objects
  - Mark all vertices as unlocked
  - **For** $i$=1 to $n/2$ **do**
    - Compute $g_{ab}$ for all pairs $a,b$
    - Pick unlocked $a_i,b_i$ with largest $g_{ab,i}$
    - Mark $a_i,b_i$ as locked
    - Store gain
    - Update $D_v$ für all objects
  - Find $k$ such that $G_k=\sum_{i=1}^{k} g_{ab,i}$ is maximal
  - **If** $G_k>0$, **then** move $a_1,\ldots,a_k$ from A to B and $b_1,\ldots,b_k$ from B to A.
- **Until** $G_k \leq 0$




- $D'_x = D_x + 2\,c_{xp} - 2\,c_{xq}$, $\forall\, x \in A - \{p\}$ (swap $p$ and $q$, $p \in A$, $q \in B$)

$$
\begin{aligned}
D'_a &= D_a + 2c_{ab} - 2c_{af} = 6 + 2 \times 1 - 2 \times 4 = 0 \\
D'_c &= D_c + 2c_{cb} - 2c_{cf} = 3 + 2 \times 1 - 2 \times 1 = 3 \\
D'_d &= D_d + 2c_{df} - 2c_{db} = 3 + 2 \times 3 - 2 \times 4 = 1 \\
D'_e &= D_e + 2c_{ef} - 2c_{eb} = 0 + 2 \times 2 - 2 \times 2 = 0
\end{aligned}
$$

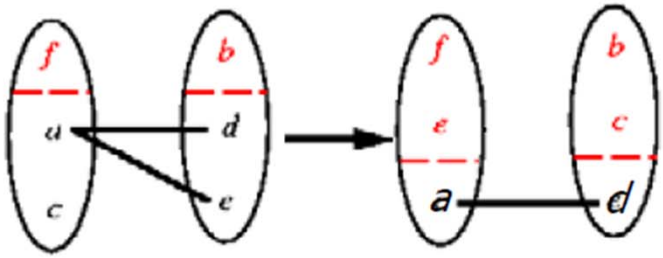
- $g_{xy} = D'_x + D'_y - 2c_{xy}$.

$$
\begin{aligned}
g_{ad} &= D'_a + D'_d - 2c_{ad} = 0 + 1 - 2 \times 3 = -5 \\
g_{ae} &= D'_a + D'_e - 2c_{ae} = 0 + 0 - 2 \times 2 = -4 \\
g_{cd} &= D'_c + D'_d - 2c_{cd} = 3 + 1 - 2 \times 3 = -2 \\
g_{ce} &= D'_c + D'_e - 2c_{ce} = 3 + 0 - 2 \times 2 = -1 \ (maximum)
\end{aligned}
$$

- Swap $c$ and $e$! $(\widehat{g_2} = -1)$

# Swapping Pair Determination

- **Repeat**
  - Compute $D_v$ für all objects
  - Mark all vertices as unlocked
  - **For** $i$=1 to $n/2$ **do**
    - Compute $g_{ab}$ for all pairs $a,b$
    - Pick unlocked $a_i,b_i$ with largest $g_{ab,i}$
    - Mark $a_i,b_i$ as locked
    - Store gain
    - Update $D_v$ für all objects
  - Find $k$ such that $G_k=\sum_{i=1}^{k} g_{ab,i}$ is maximal
  - **If** $G_k>0$, **then** move $a_1,\ldots,a_k$ from A to B and $b_1,\ldots,b_k$ from B to A.
- **Until** $G_k \leq 0$




- $D''_x = D'_x + 2\,c_{xp} - 2\,c_{xq},\ \forall\, x \in A - \{p\}$

$$D''_a = D'_a + 2c_{ac} - 2c_{ae} = 0 + 2 \times 2 - 2 \times 2 = 0$$
$$D''_d = D'_d + 2c_{de} - 2c_{dc} = 1 + 2 \times 4 - 2 \times 3 = 3$$

- $g_{xy} = D''_x + D''_y - 2c_{xy}$.

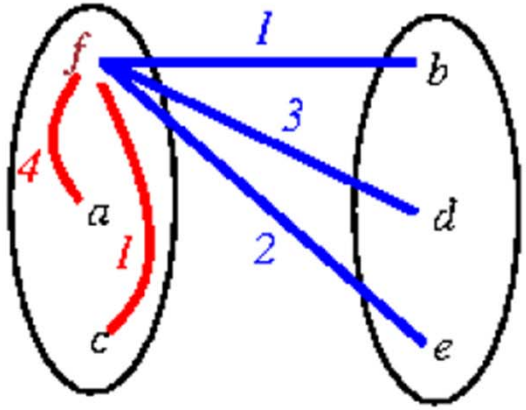
$$g_{ad} = D''_a + D''_d - 2c_{ad} = 0 + 3 - 2 \times 3 = -3 (\hat{g_3} = -3)$$

- Note that this step is redundant $(\sum_{i=1}^{n} \hat{g_i} = 0)$.
- Summary: $\hat{g_1} = g_{bf} = 4,\ \hat{g_2} = g_{ce} = -1,\ \hat{g_3} = g_{ad} = -3.$
- Largest partial sum $\max \sum_{i=1}^{k} \hat{g_i} = 4$ $(k = 1) \Rightarrow$ Swap $b$ and $f$.

# Next Iteration

- **Repeat**
  - Compute $D_v$ für all objects
  - Mark all vertices as unlocked
  - **For** $i$=1 to $n/2$ **do**
    - Compute $g_{ab}$ for all pairs $a,b$
    - Pick unlocked $a_i,b_i$ with largest $g_{ab,i}$
    - Mark $a_i,b_i$ as locked
    - Store gain
    - Update $D_v$ für all objects
  - Find $k$ such that $G_k=\sum_{i=1}^{k} g_{ab,i}$ is maximal
  - **If** $G_k>0$, **then** move $a_1,\ldots,a_k$ from A to B and $b_1,\ldots,b_k$ from B to A.
- **Until** $G_k \leq 0$

| | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| a | 0 | 1 | 2 | 3 | 2 | 4 |
| b | 1 | 0 | 1 | 4 | 2 | 1 |
| c | 2 | 1 | 0 | 3 | 2 | 1 |
| d | 3 | 4 | 3 | 0 | 4 | 3 |
| e | 2 | 2 | 2 | 4 | 0 | 2 |
| f | 4 | 1 | 1 | 3 | 2 | 0 |




*Initial cut cost* = (1+3+2)+(1+3+2)+(1+3+2) = 18 (22−4)

- Iteration 2: Repeat what we did at Iteration 1 (Initial cost = 22-4 =18).
- Summary: $\hat{g_1} = g_{ce} = -1,\ \hat{g_2} = g_{ab} = -3,\ \hat{g_3} = g_{fd} = 4.$
- Largest partial sum = $\max \sum_{i=1}^{k} \hat{g_i} = 0$ ($k$=3) $\Rightarrow$ Stop!

# Simulated Annealing

- General method for solving combinatorial optimization problems.
- Based the model of slowly cooling crystal liquids.
- Changes leading to a poorer configuration (with respect to some cost function) are accepted with a certain probability.
- This probability is controlled by a temperature parameter: the probability is smaller for smaller temperatures.

# Simulated Annealing Algorithm

```
procedure SimulatedAnnealing;
var i, T: integer;
 begin
   temp := temp_start;
   cost:=c(P);
   while (Frozen()==FALSE) do
    begin
     while (Equilibrium()==FALSE) do
       begin P' := RandomMove(P);
         cost'=c(P')
         deltacost := cost' - cost;
         if (Accept(deltacost, temp)>random[0,1))
         then P=P'; cost=cost'
       end;
      temp:= decreaseTemp(temp)
   end;
end;
```

# Simulated Annealing

- Annealing schedule: DecreaseTemp(), Frozen()
  - temp_start=1.0
  - temp = $\alpha \cdot$ temp (typical: $0.8 \leq \alpha \leq 0.99$)
  - stop at temp < temp_min or if no more improvement
- Equilibrium:
  - After certain number of iterations or when no more improvement
- Complexity:
  - From exponential to constant, depending on choice of Equilibrium(), DecreaseTemp(), Frozen()
  - The longer the runtime, the better the results
  - Usually functions constructed to obtain polynomial runtime

# And more ...

MSc THESIS, Roel Meeuws, 2007, Delft

| Paper | Dynamic/ Static | Strategy | Criteria | Model/ Data Structure | Granularity of Partitioning | Time Complexity |
|---|---|---|---|---|---|---|
| [58] | Static | Simulated Annealing | n/a | n/a | n/a | n/a |
| [42] | Static | Greedy | Minimal area, data-rate constraints | System Graph Model (like H-CDFG) | operations | linear |
| [41] | Static | Greedy (see [42]) | Minimal area, data-rate constraints | Hierarchical Sequence Graph | operations | n/a |
| [77] | Static | Simulated Annealing | Minimal communication cost | Petri-nets, (annotated) CDFG | operations | $O(tn)$ t=temperature steps |
| [34] | Static | Simulated Annealing | Hardware suitability (compare local phase [54]) | (extended) $C^X$ syntax graph | basic blocks | n/a |
| [54] | Static | GCLP | GC objective function (e.g. Area combined with speed) | n/a | Tasks (instruction level subgraphs) | $O(ne)$, $e$=edges |
| [96] | Static | Binary Constraint Search | Constraints of encapsulated partitioning algorithm | n/a | n/a | $O(part(S))$ $part(S)$ = encaps. part. alg. |
| [50] | Static | Dynamic Programming | Temporal size of loops / leaf functions | n/a | loops, leaf functions | n/a |
| [53] | Static | GCLP (MIBS) | See [54] | CDFG | Tasks | $O(n^3 + n^2B)$, $B$=bins |
| [82] | Static | Evolutionary (Genetic) | Minimal area, timing and concurrency constraints | CDFG | functional elements | $O(gp)$, $g$=generations, $p$=population |
| [30] | Static | Clustering | Minimal cost, minimal power, timing and power constraints | Task Graph | task clusters | n/a |
| [29] | Dynamic | Greedy, Clustering | Minimize area, timing constraints | Task Graph | task clusters | n/a |
| [65] | Dynamic | Clustering | Area constraints | CDFG | loop clusters | linear |
| [89] | Static | Evolutionary (Genetic) | maximize fitness (minimize area and interconnect) | DFG | fine:operations coarse:DFGs | n/a |
| [84] | Dynamic | Evolutionary | Maximum rank (Pareto ranking in power and price) | Task Graph | Tasks | n/a |
| [91] | Static | Greedy | Temporal size of loops / leaf functions | n/a | loops | n/a |
| [14] | Static | Dynamic Programming | Minimum latency, resource constraints | DFG | Tasks | polynomial |
| [12] | Static | Simulated Annealing, Kernighan-Lin | Minimize latency, area constraints | Call graph | functions | n/a |

Table 2.1: Inventarization of several papers on hardware software partitioning with corresponding partitioning schemes, criteria, and data structures

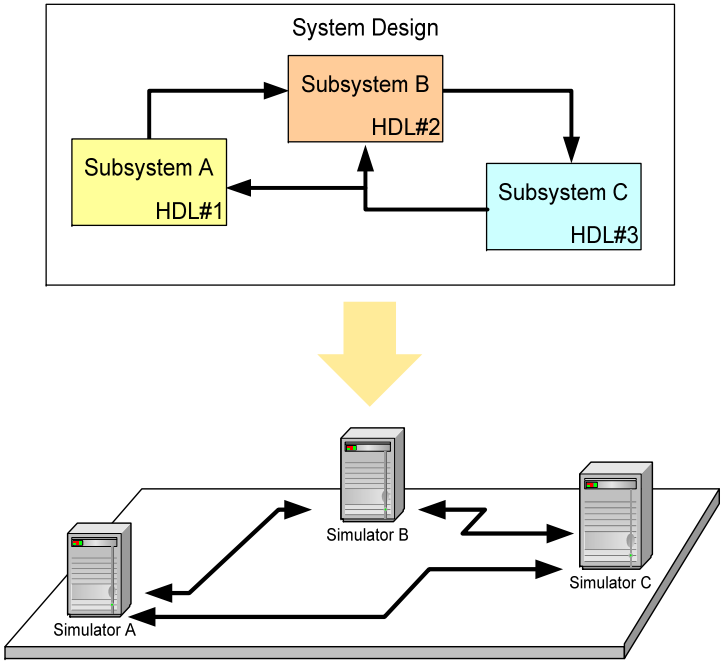
# HW/SW Co-Simulation

System Architect Designer


# Introduction: Co-Simulation

Co-Simulation:

- Simulation methodology
- Individual components simulated by different simulation tools
- Different modeling languages
- Different abstraction levels
- But: common co-simulation




Why use co-simulation?

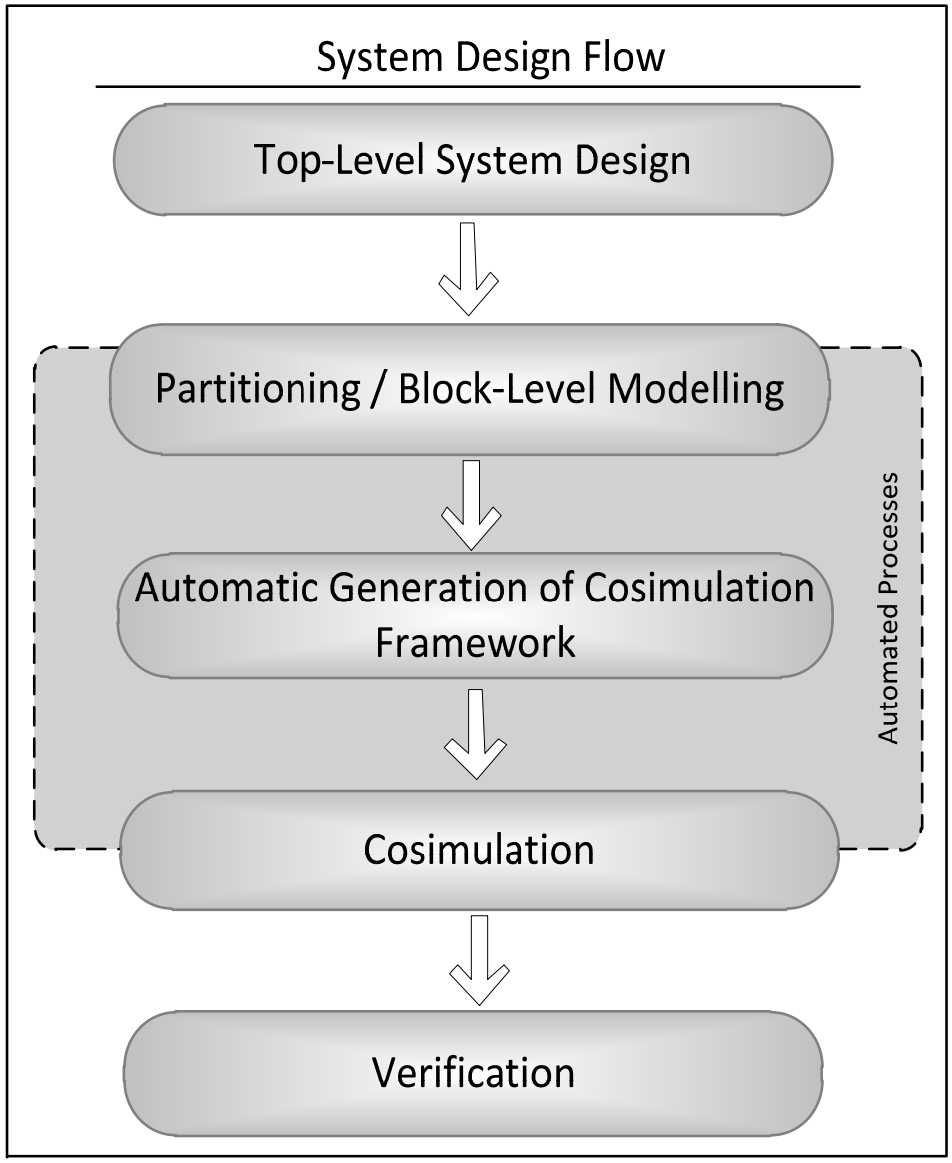
- Handling increased complexity
- Flexibility
- Verification already in early design phases
- Simulation performance improvements
- Short development cycles

# System Architect Designer SyAD

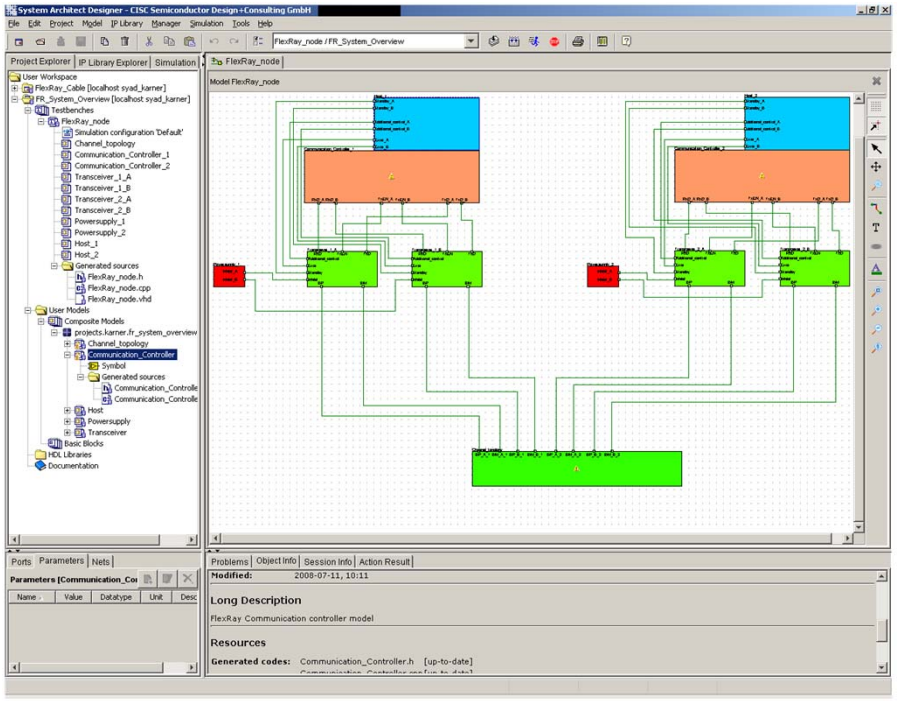
Multi-HDL design

- SystemC [SystemC]
- ModelSim [VHDL]
- ADVanceMS [VHDL/AMS]
- NCSIM-SimVision (AMS Designer) [VHDL/AMS]
- Saber [SaberMAST]
- Simulink [Matlab/Simulink]

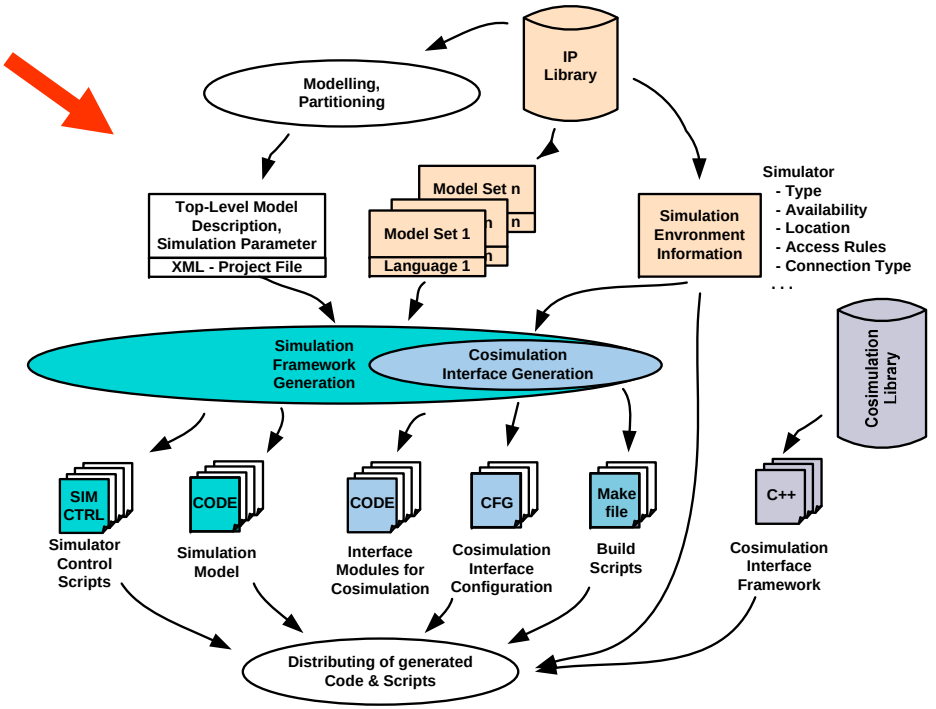
# Design Methodology



# SyAD: Co-Simulation




**Synchronisation method is implemented as decentralized, “synchronous”, conservative protocol**



# Motivation for Run-Time Co-Simulation Model Switching

## System level:

- Validation and analysis of entire embedded systems
- Focus: short simulation time for longer simulated time (>> 1s)
- Abstracted behavior: hides low-level effects that might propagate

## Physical level:

- High simulation time: simulation of complex analog components
- Relatively short simulated times (µs, ms)
- Detailed behavior

## Co-simulation problems:

- Simulated times: physical level vs. system level
- Co-simulation performance: determined by slowest simulator
  → critical in physical level/system level co-simulation

**Idea: Run-time switching of co-simulation models**

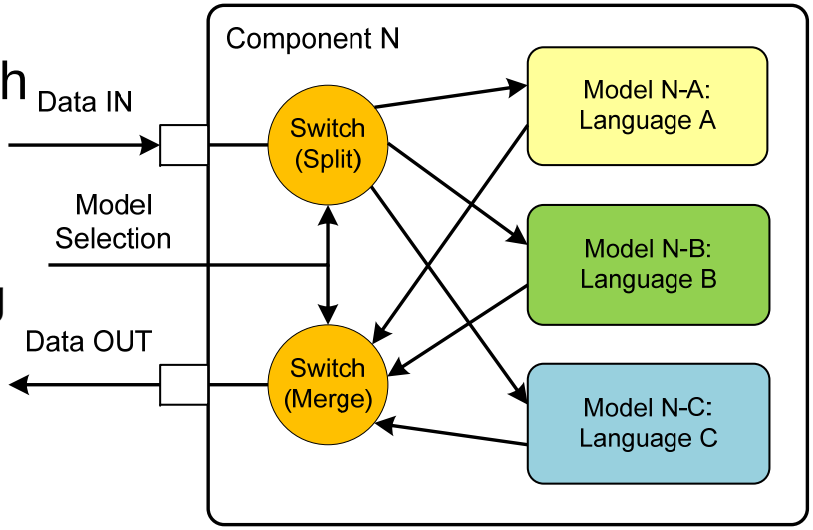
# Run-Time Co-Simulation Model Switching

**Run-time co-simulation model switching:**

- Modeling of a single component by using multiple HDL (discrete & continuous) and abstraction levels
- Synchronized run-time switching between the abstraction level models

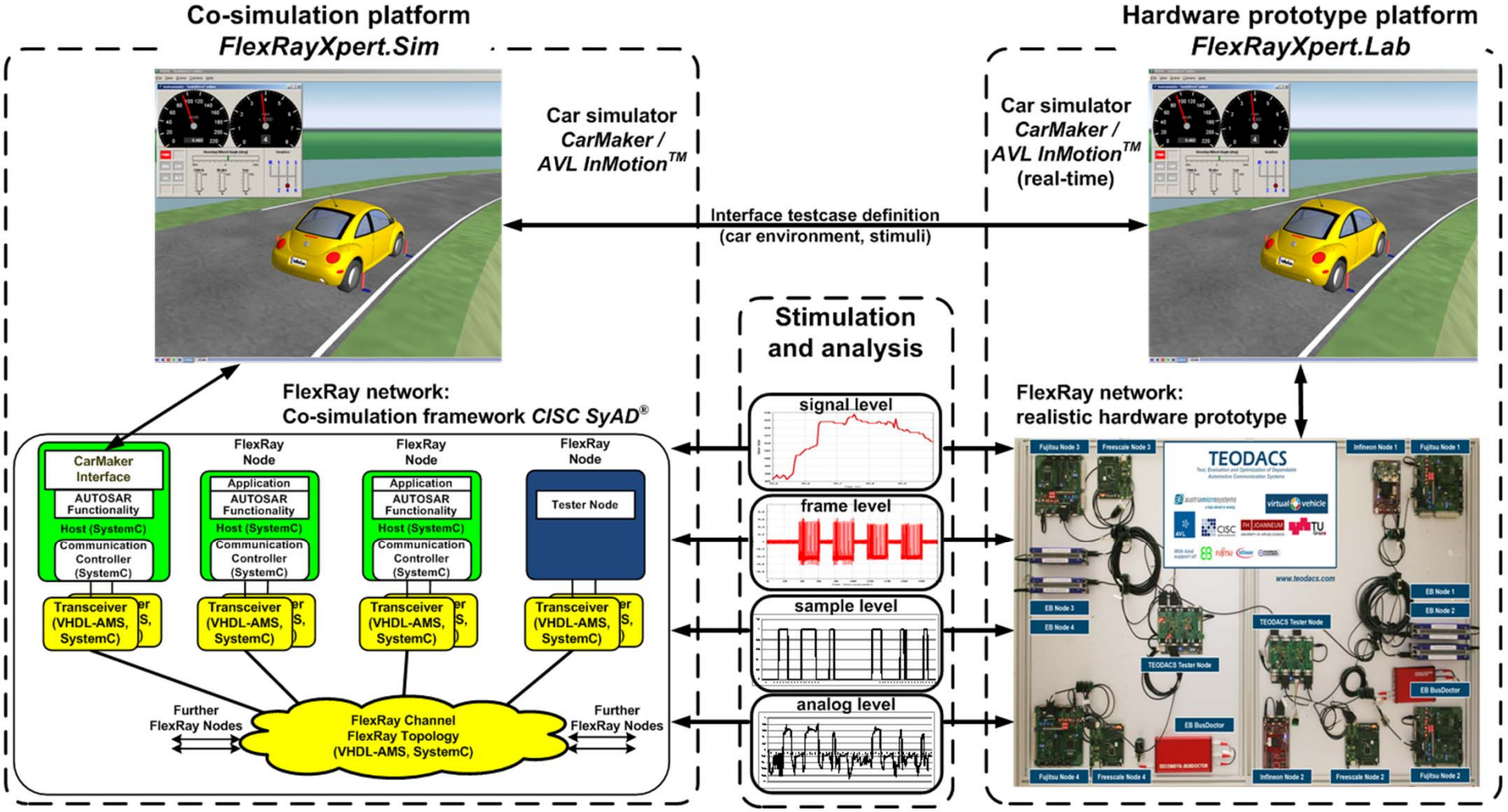
**Features:**

- Long simulated time / high simulation speed (system level models) plus high accuracy (low physical level models)
  - Using fast high level models during normal circumstances
  - Switch to high-detailed models during time intervals of particular interest
- Enhances co-simulation speed
  - Using computational expensive simulation models only in a clearly defined area

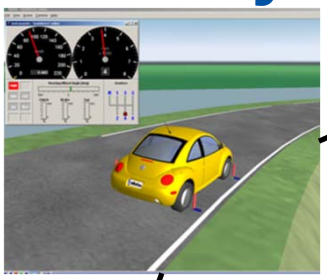


# TEODACS: Overview

**TEODACS:** **T**est, **E**valuation and **O**ptimization of **D**ependable **A**utomotive **C**ommunication **S**ystems



# FlexRayXpert.Sim: Experimental Setup



FlexRay Node 2
Application
Communication Controller 2 (SystemC)
T … FlexRay Termination
1.0m … Cable Segment Length
SWITCH 2
1: VHDL-AMS Topology Model
Transceiver (VHDL-AMS)
0.4m
Transceiver (VHDL-AMS)
2.0m
2.0m
1.0m
Transceiver (VHDL-AMS)
2.0m
0.5m
0.8m (7.0m)
Transceiver (VHDL-AMS)
Transceiver (VHDL-AMS)
FlexRay Node 0
Application
Communication Controller 0 (SystemC)
SWITCH 0
SWITCH 4
FlexRay Node 4
Application
Communication Controller 4 (SystemC)
2: SystemC Topology Model
Transceiver (SystemC)
0.4m
Transceiver (SystemC)
2.0m
2.0m
1.0m
Transceiver (SystemC)
2.0m
0.5m
0.8m (7.0m)
Transceiver (SystemC)
Transceiver (SystemC)
SWITCH 1
SWITCH 3
FlexRay Node 1
Communication Controller 1 (SystemC)
Application
Communication Controller 3 (SystemC)
Application
FlexRay Node 3

# PowerCard - Methodologies for Designing Power-Aware Smart Card Systems

# Contactless Smart Cards as Future Mobile Devices

- Contactless smart card controllers are currently used in various demanding applications
  - payment, e.g. debit/credit cards
  - identification, e.g. electronic passport
  - pay TV

- …and there exist ideas for much more complex use cases by connecting displays, buttons, and finger print sensors to the controller


smart card with OLED display (Samsung, 2010)


smart card with numeric key pad and display (NagraID, 2010)

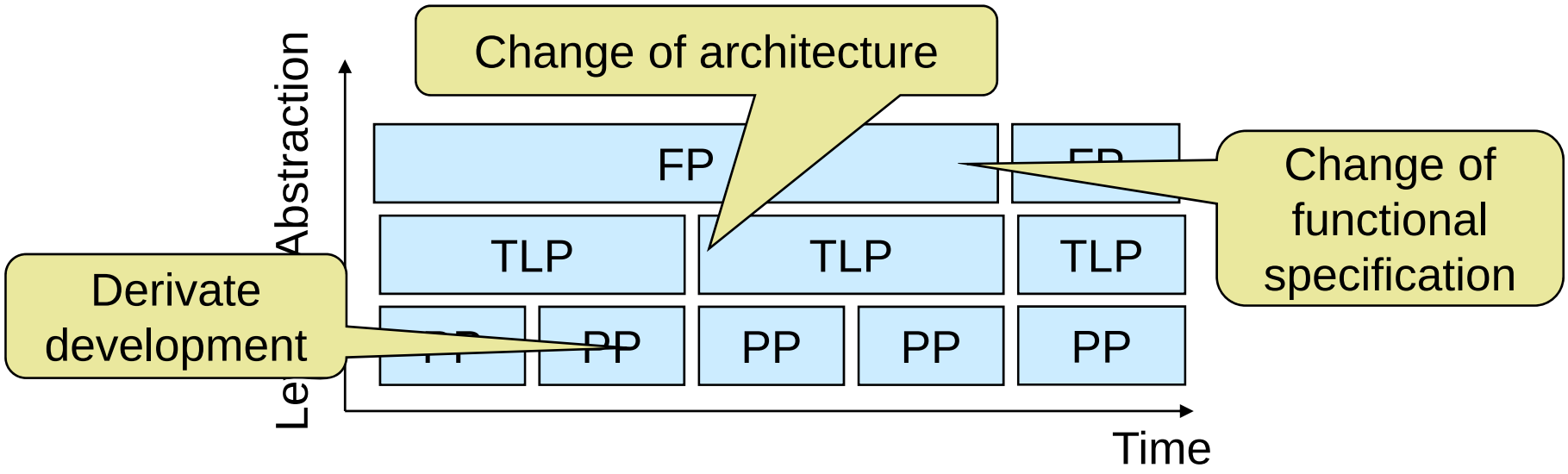
# System Abstraction

- Requirements
  - In general independent of hardware and software
  - Basic smart card OS functionality should be provided
  - Focus on algorithm design and memory system (limited resource)
- Levels of Abstraction

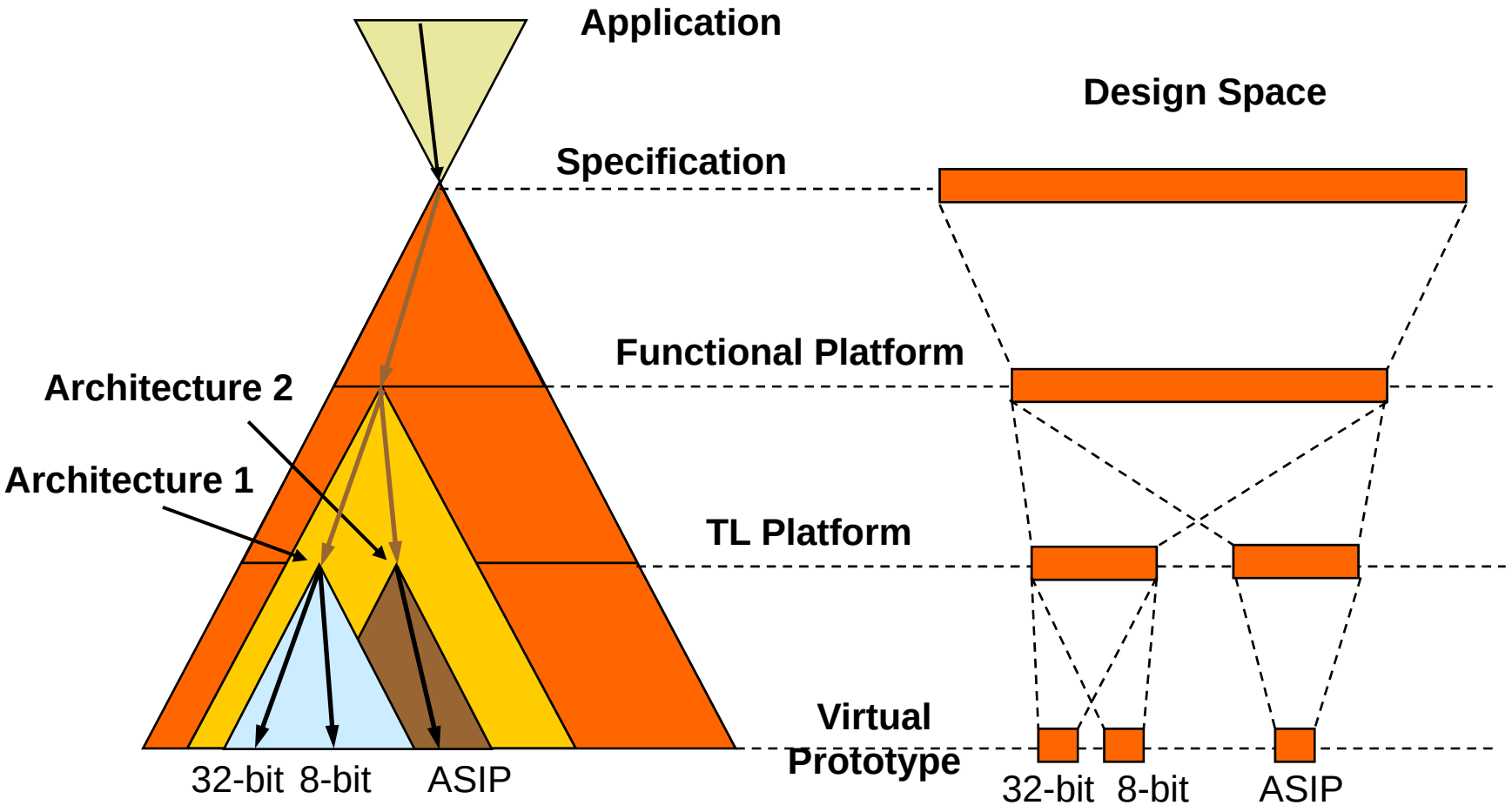
| | Model representation | Model of computation |
|---|---|---|
| Functional Level | Object oriented<br>Interface-based commun. | Sequential execution<br>Untimed/timed |
| Transaction Level | TL1 SoC Model<br>Abstract processing units | SoC: Parallel tasks, timed<br>SW: untimed, delays |
| Prototype Level | Cycle accurate HW<br>Cross-compiled software | Parallel hw models, FSMs<br>Sequential software |

# Platform Lifetime

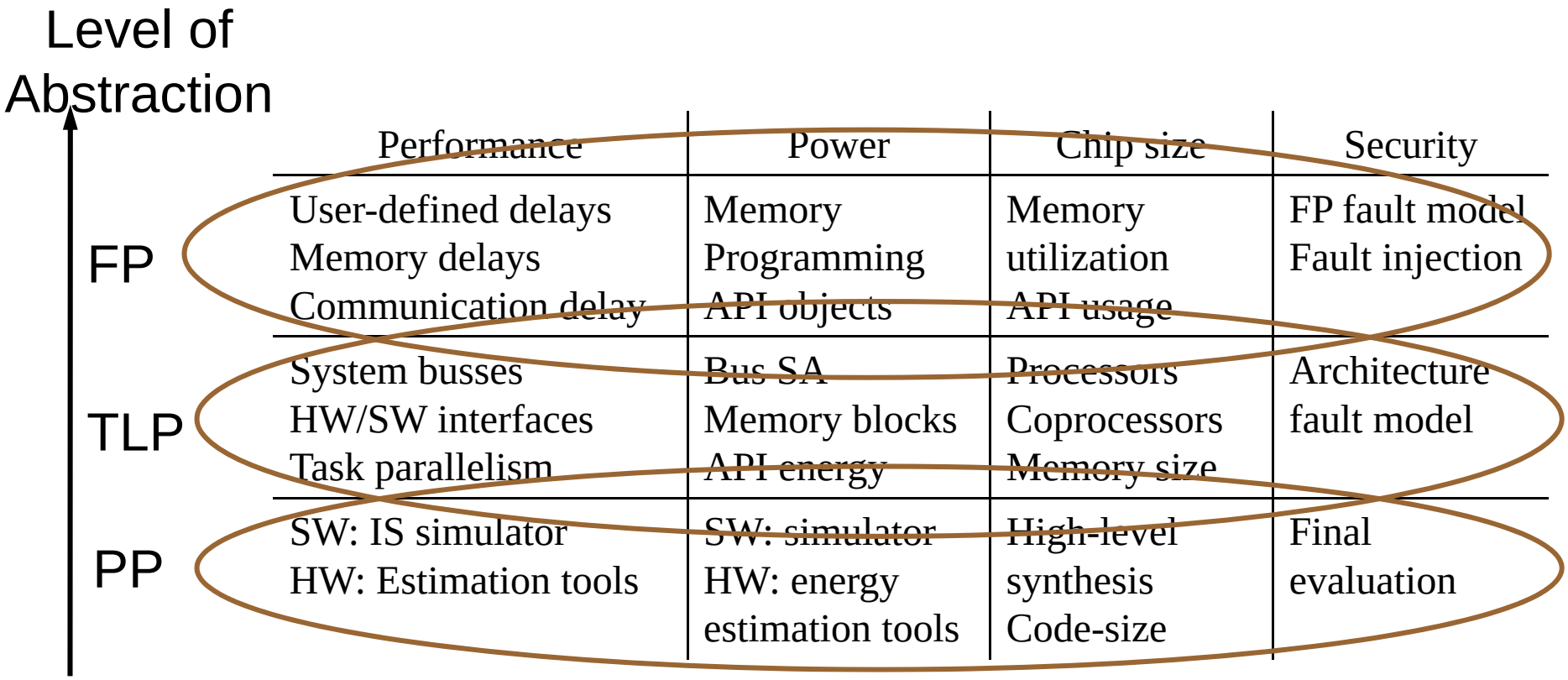
- Abstract platforms are more stable
- Different solutions can be derived from an abstract model
- This results in more stable systems than old system redesign



# Design space Exploration based on hierarchical platforms

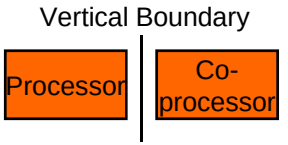


# Design Space Examination

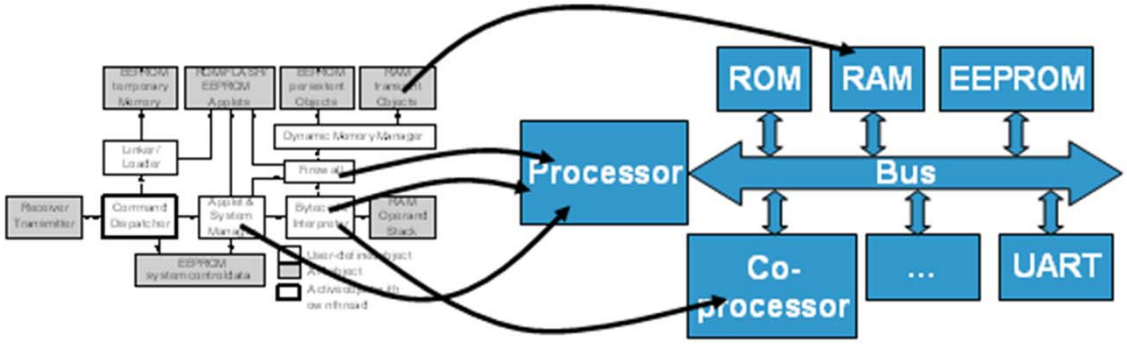



| Level of Abstraction | Performance | Power | Chip size | Security |
|---|---|---|---|---|
| FP | User-defined delays<br>Memory delays<br>Communication delay | Memory<br>Programming<br>API objects | Memory<br>utilization<br>API usage | FP fault model<br>Fault injection |
| TLP | System busses<br>HW/SW interfaces<br>Task parallelism | Bus SA<br>Memory blocks<br>API energy | Processors<br>Coprocessors<br>Memory size | Architecture<br>fault model |
| PP | SW: IS simulator<br>HW: Estimation tools | SW: simulator<br>HW: energy<br>estimation tools | High-level<br>synthesis<br>Code-size | Final<br>evaluation |

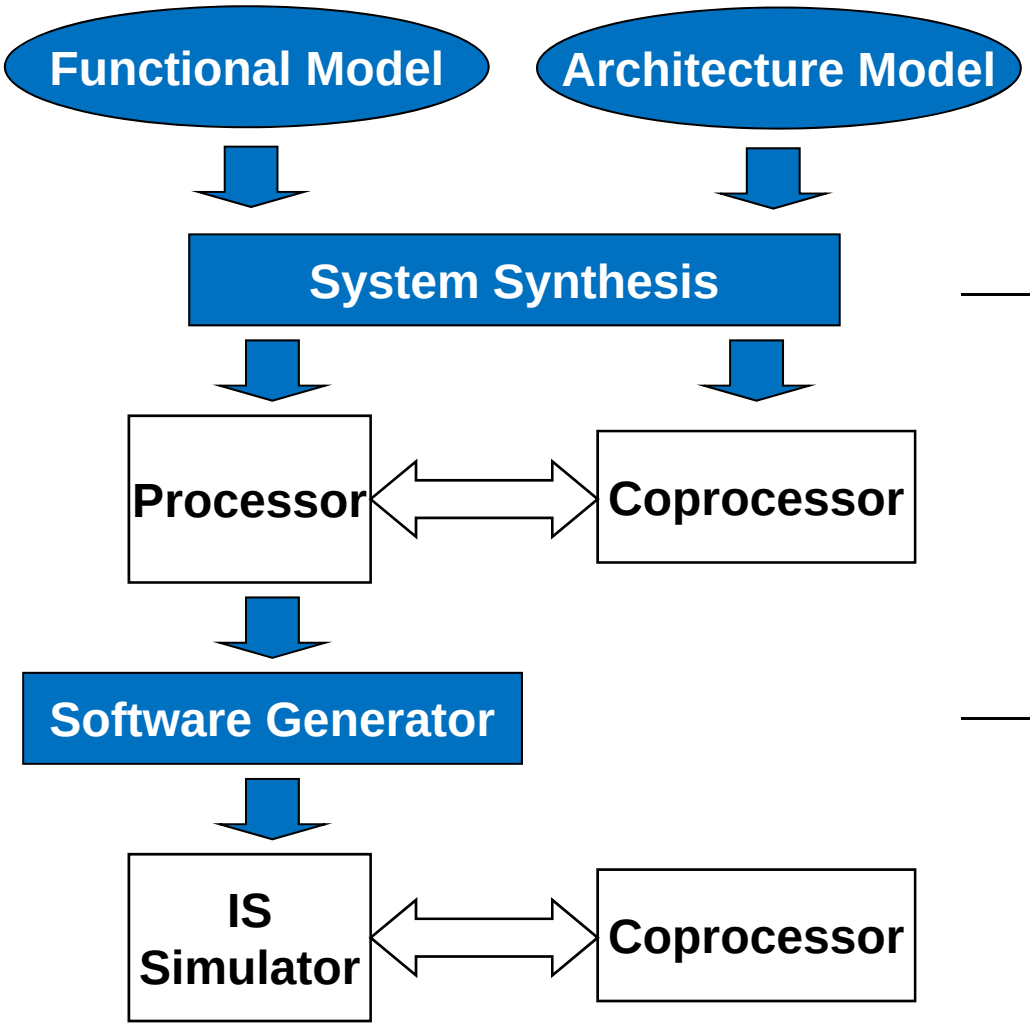
# Vertical Codesign




- Target Architecture
  - Existing processor platform
  - HW acceleration based on instruction-set extension and coprocessor
- Codesign Approach
  - Evaluation of different configurations
  - Optimization of the HW/SW interface
  - Cosimulation comprising hardware, all software layers and application



# Vertical Codesign




Functional Platform Model:
-Interacting C++ objects
-SystemC simulation kernel
-Includes: Application, OS, HW

---

Transaction-level Model:
-HW/SW Mapping
-HW/SW Interface Optimization
-Memory access optimization
-Memory system design

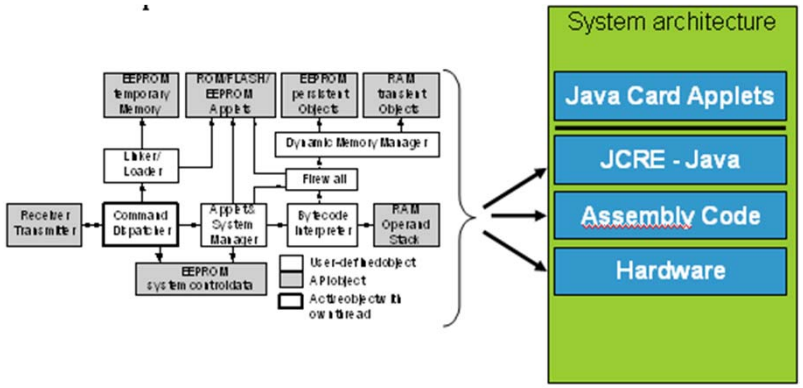
---

Prototype Platform:
-Software design
-Software power estimation
-Software power optimization
-Memory access optimization

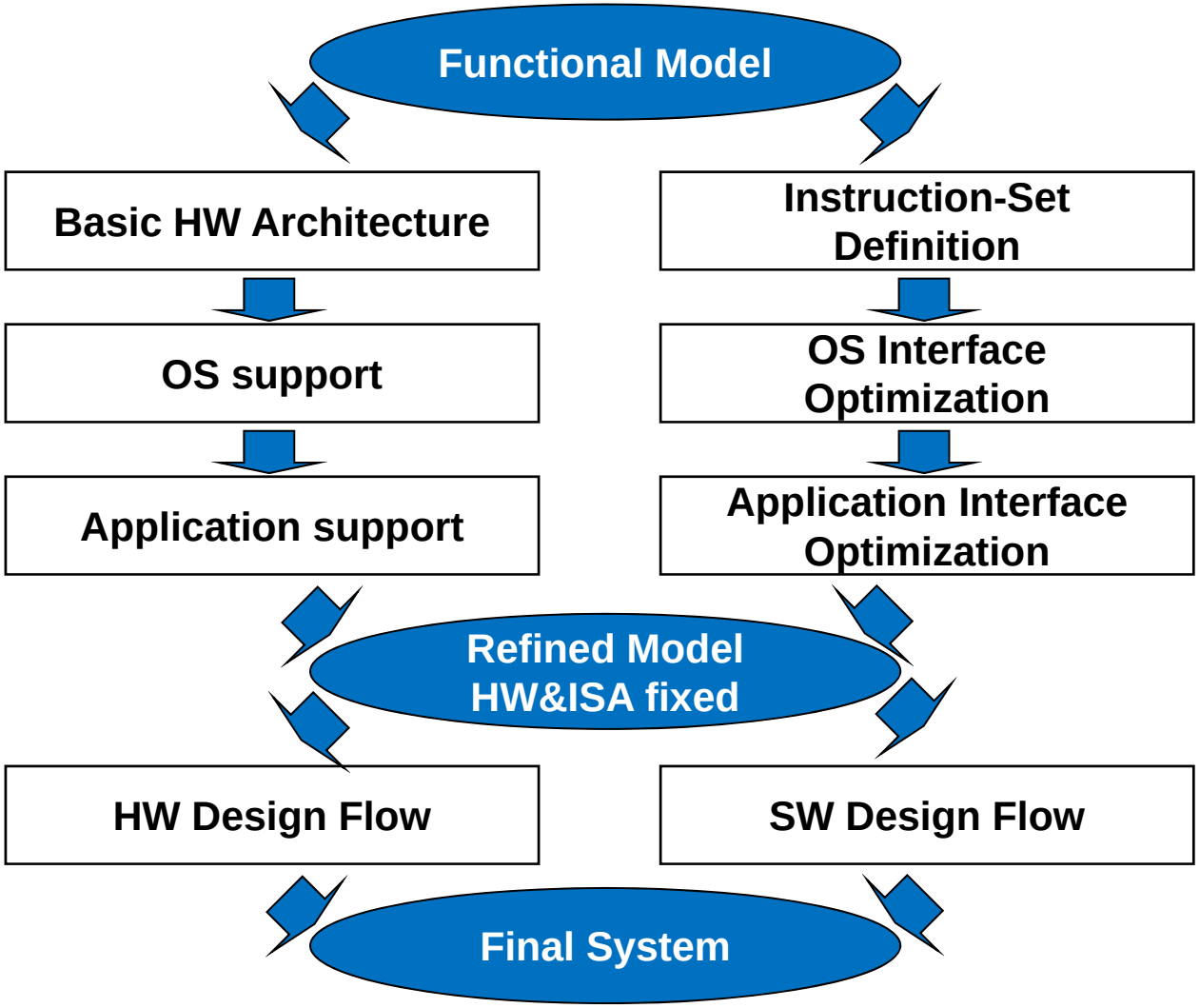
# Horizontal Codesign


- Target Architecture
  - New hardware components, application specific instruction-set processors
  - Optimized hardware for a dedicated application

- Codesign Approach
  - Design of hardware and software layers with regard to the target application
  - Stepwise refinement and cosimulation



# Horizontal Codesign

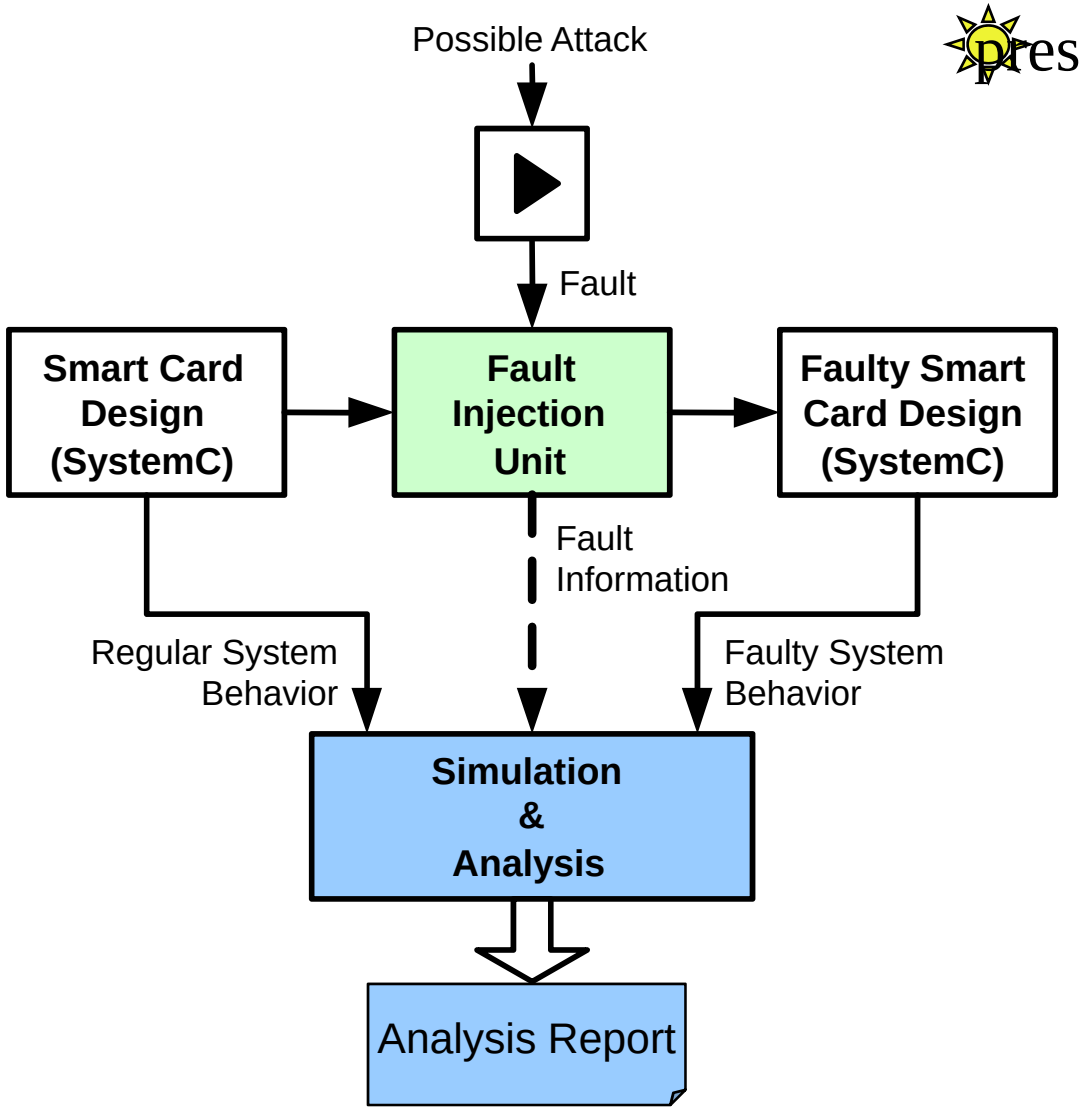


# Design Flow with Security Extension based on Power Profile

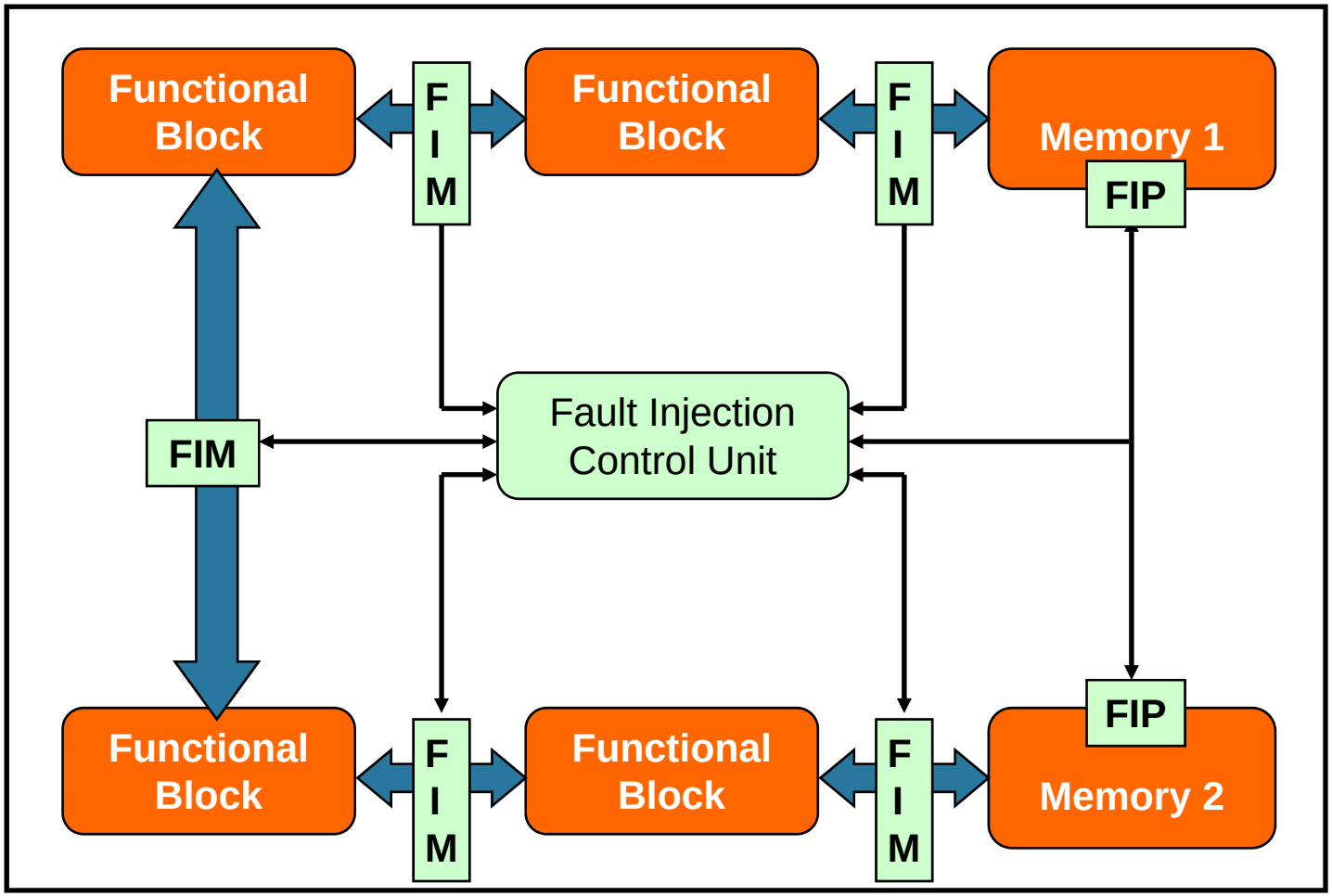
- Smart cards store and deal with sensitive data
  - SIM cards in mobile phones
  - e-purse
  - contact-less ID systems
- Security attacks on smart cards
  - invasive or semi-invasive attacks
- Test robustness against attacks
- Attack simulation early in the design process using fault injection
  - ease design changes and
  - insertion of protection mechanisms
- SystemC for high simulation performance
  - can be applied on all SystemC designs

# Attack Simulation Flow

Presented at ATS'04



# Fault Injection in Functional Design



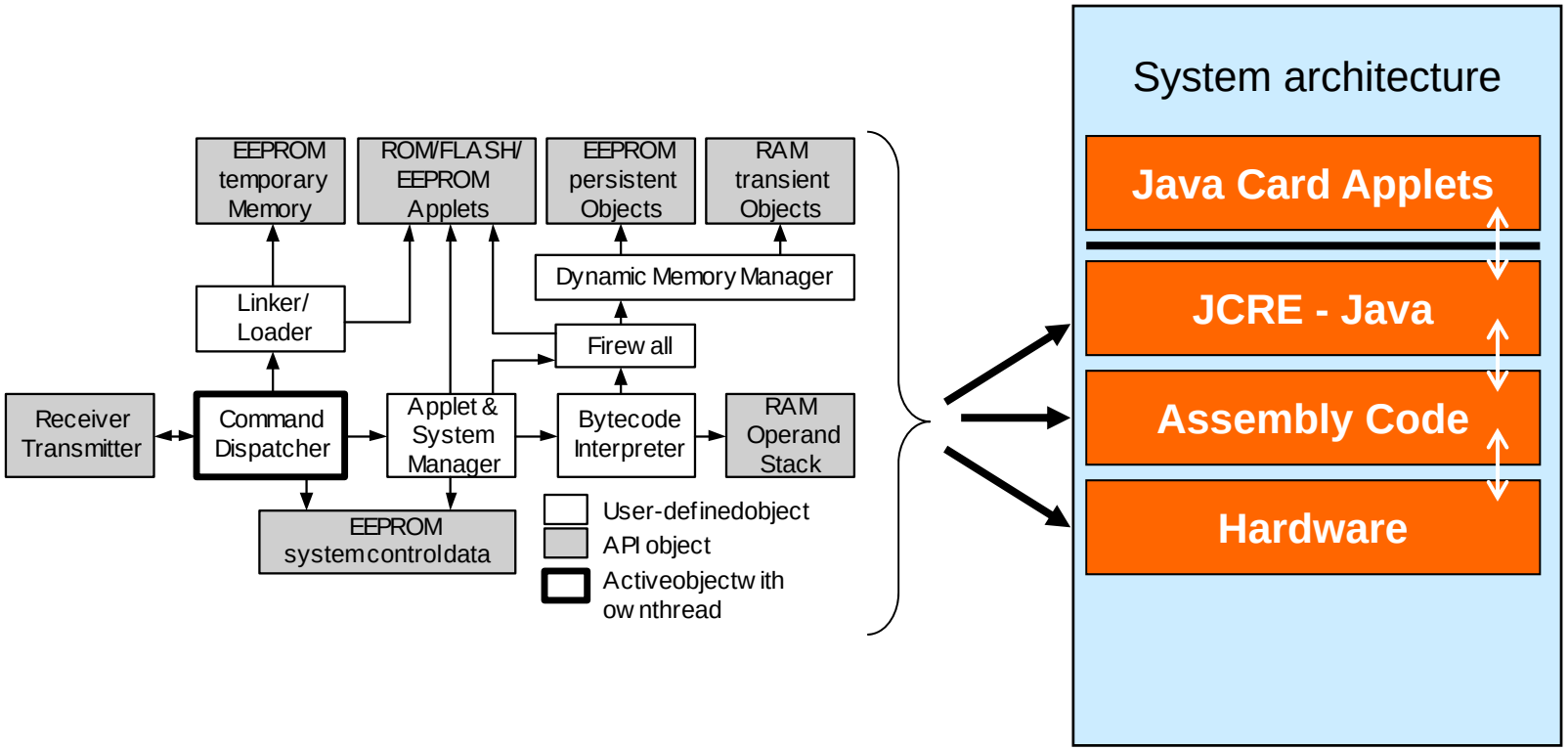

FIM … Fault Injection Module
FIP … Fault Injection Port

# Methodology Evaluation

- Evaluation with a Java Card™ Virtual Machine Implementation
- Evaluation Steps:
  - Implementation JCVM functional platform model
  - Vertical Codesign
    - 32-bit Solution based on MIPS Architecture
    - 8-bit Solution based on 8051 Architecture
  - Horizontal Codesign
    - Application Specific Instruction-set Processor

# Horizontal Codesign Solutions

- Vertical integration of functional units
- Model comprises virtual machine as well as JC runtime



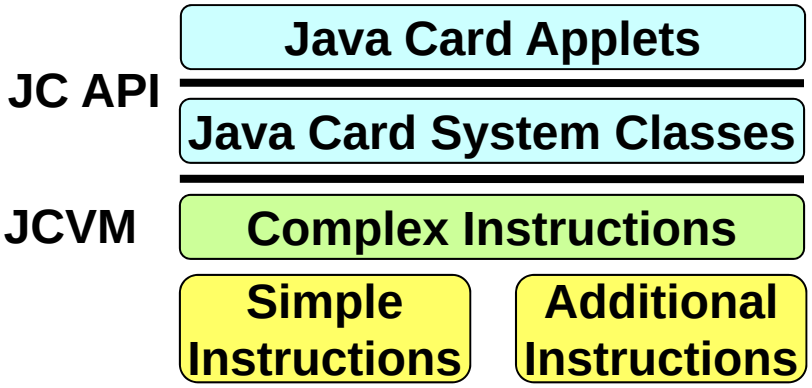
# JAVA Card ASIP Concept

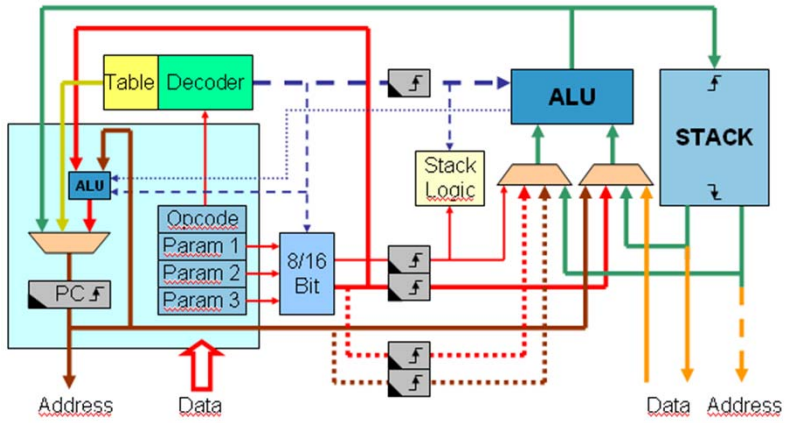
3 Classes of instructions:

- simple byte codes
- instruction set extension
- complex instructions

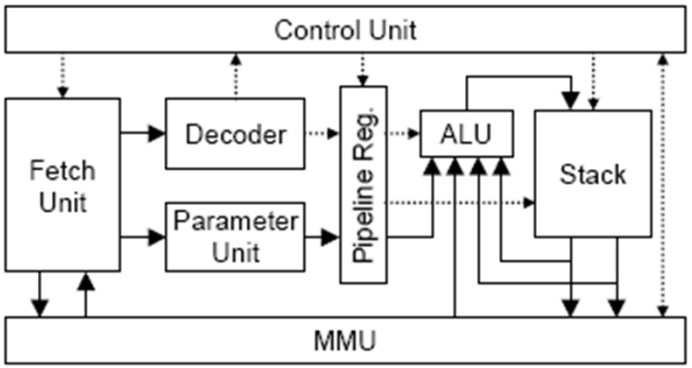
Security concept:

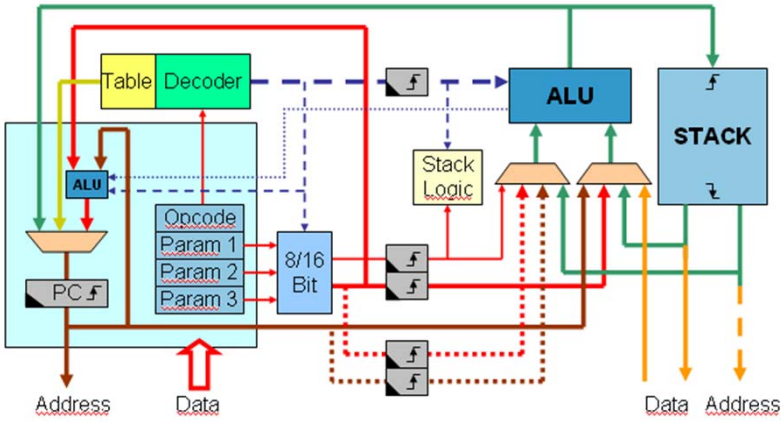
- User and kernel mode
- different instructions for different memory areas
- large MMU



# JAVA Card ASIP Architecture




| | | | | | |
|---|---|---|---|---|---|
| NOP | SALOAD | SDIV | IF_SCMPLT | PUTFIELD_S | IF_SCMPEQ_W |
| ACONST_NULL | ASTORE | SREM | IF_SCMPGE | INVOKEVIRTUAL | IF_SCMPNE_W |
| SCONST_M1 | SATORE | SNEG | IF_SCMPGT | INVOKESPECIAL | IF_SCMPLT_W |
| SCONST_0 | ASTORE_0 | SSHL | IF_SCMPLE | INVOKESTATIC | IF_SCMPGE_W |
| SCONST_1 | ASTORE_1 | SSHR | GOTO | INVOKEINTERFACE | IF_SCMPGT_W |
| SCONST_2 | ASTORE_2 | SUSHR | JSR | NEW | IF_SCMPLE_W |
| SCONST_3 | ASTORE_3 | SAND | RET | NEWARRAY | GOTO_W |
| SCONST_4 | SSTORE_0 | SOR | STABLESWITCH | ANEWARRAY | GETFIELD_A_W |
| SCONST_5 | SSTORE_1 | SXOR | SLOOKUPSWITCH | ARRAYLENGTH | GETFIELD_B_W |
| BSPUSH | SSTORE_2 | SINC | ARETURN | ATHROW | GETFIELD_S_W |
| SSPUSH | SSTORE_3 | S2B | SRETURN | CHECKCAST | GETFIELD_A_THIS |
| ALOAD | AASTORE | IFEQ | RETURN | INSTANCEOF | GETFIELD_B_THIS |
| SLOAD | BASTORE | IFNE | GETSTATIC_A | SINC_W | GETFIELD_S_THIS |
| ALOAD_0 | SASTORE | IFLT | GETSTATIC_B | IFEQ_W | PUTFIELD_A_W |
| ALOAD_1 | POP | IFGE | GETSTATIC_S | IFNE_W | PUTFIELD_B_W |
| ALOAD_2 | POP2 | IFGT | PUTSTATIC_A | IFLT_W | PUTFIELD_S_W |
| ALOAD_3 | DUP | IFLE | PUTSTATIC_B | IFGE_W | PUTFIELD_A_THIS |
| SLOAD_0 | DUP2 | IFNULL | PUTSTATIC_S | IFGT_W | PUTFIELD_B_THIS |
| SLOAD_1 | DUP_X | IFNONNULL | GETFIELD_A | IFLE_W | PUTFIELD_S_THIS |
| SLOAD_2 | SWAP_X | IF_ACMPEQ | GETFIELD_B | IFNULL_W | |
| SLOAD_3 | SADD | IF_ACMPNE | GETFIELD_S | IFNONNULL_W | |
| AALOAD | SSUB | IF_SCMPEQ | PUTFIELD_A | IF_ACMPEQ_W | |
| BALOAD | SMUL | IF_SCMPNE | PUTFIELD_B | IF_ACMPNE_W | |

## Performance comparison

**MIPS**
asReturn: 74 bytes
popFrame: 80 bytes
pushFrame: 164 bytes

**ASIP**
asReturn: 10 bytes
popFrame: 20 bytes
pushFrame: 129 bytes

### Interpretation

- Code density is much higher
- Microcoded routines run therefor faster (less instructions)
- Specialized hardware gives additional performance boost