BCS APG 10.10.13

System software support of hardware efficiency

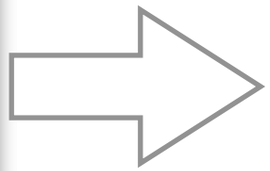
by Igor Schagaev

©

Plan for today - I

Theories in brief:

- FT: GAFT, Processes
- Redundancy
- Recoverability
- Reconfigurability

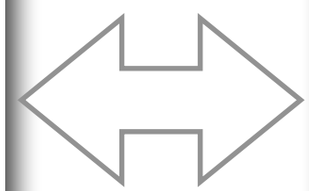


Properties...

- Reliability...
- Fault tolerance...
- Performance...
- Maintainability...
- Adaptability...
- Scalability...
- Life circle costs (manufacturing, run-time, utilization)
- Energy efficiency
- Ease of use (learning, application, maintenance)

System Structure Change

- GAFT, Supportive processes
- Redundancy handling
- Reconfigurability support
- Static&Dynamic Control of RP



Plan for today -11

System Software for FT: Language

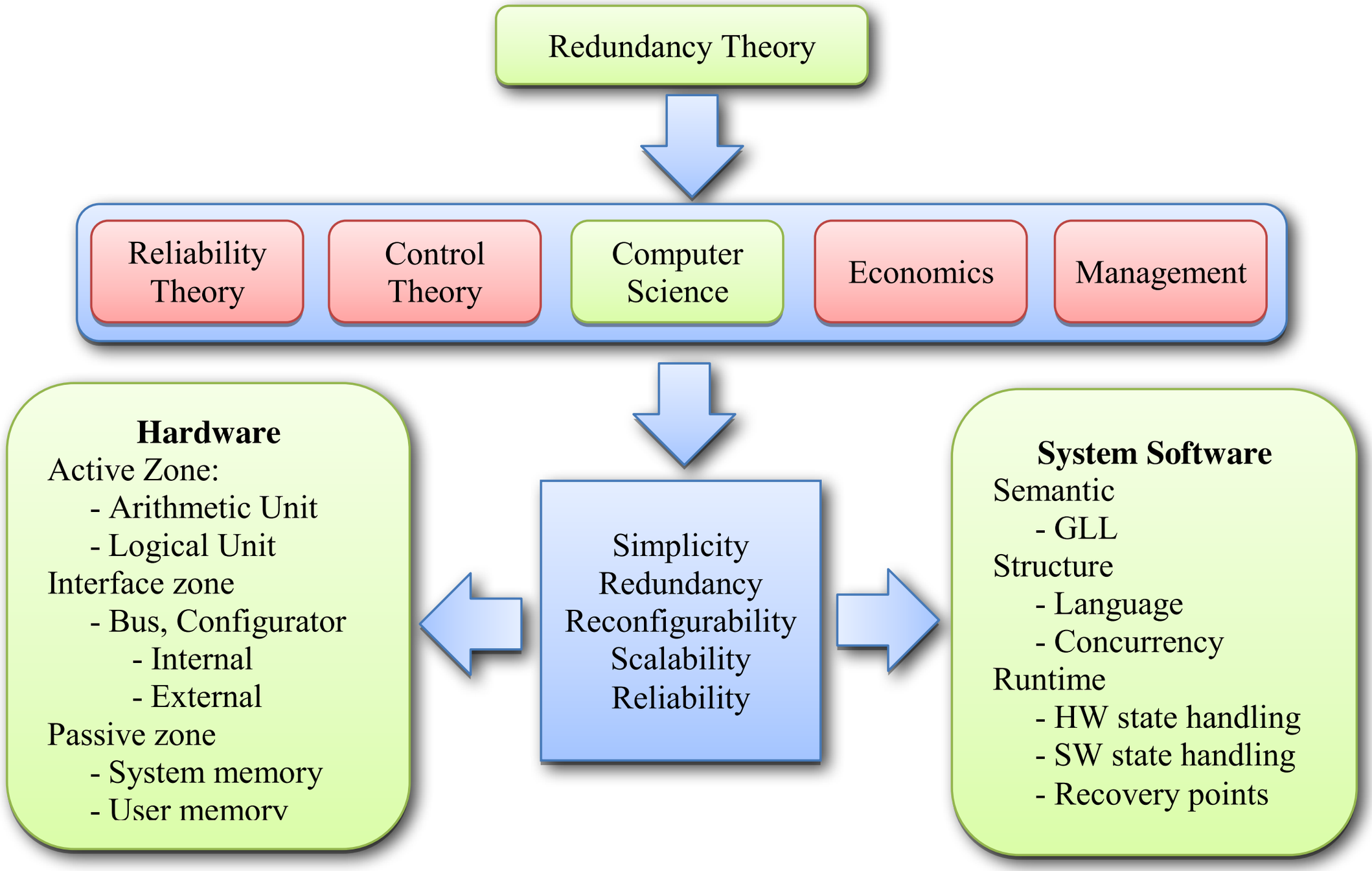
- Data structures
- Control operators
- Semaphores
- Recovery point formation

System Software for FT: Run-time system

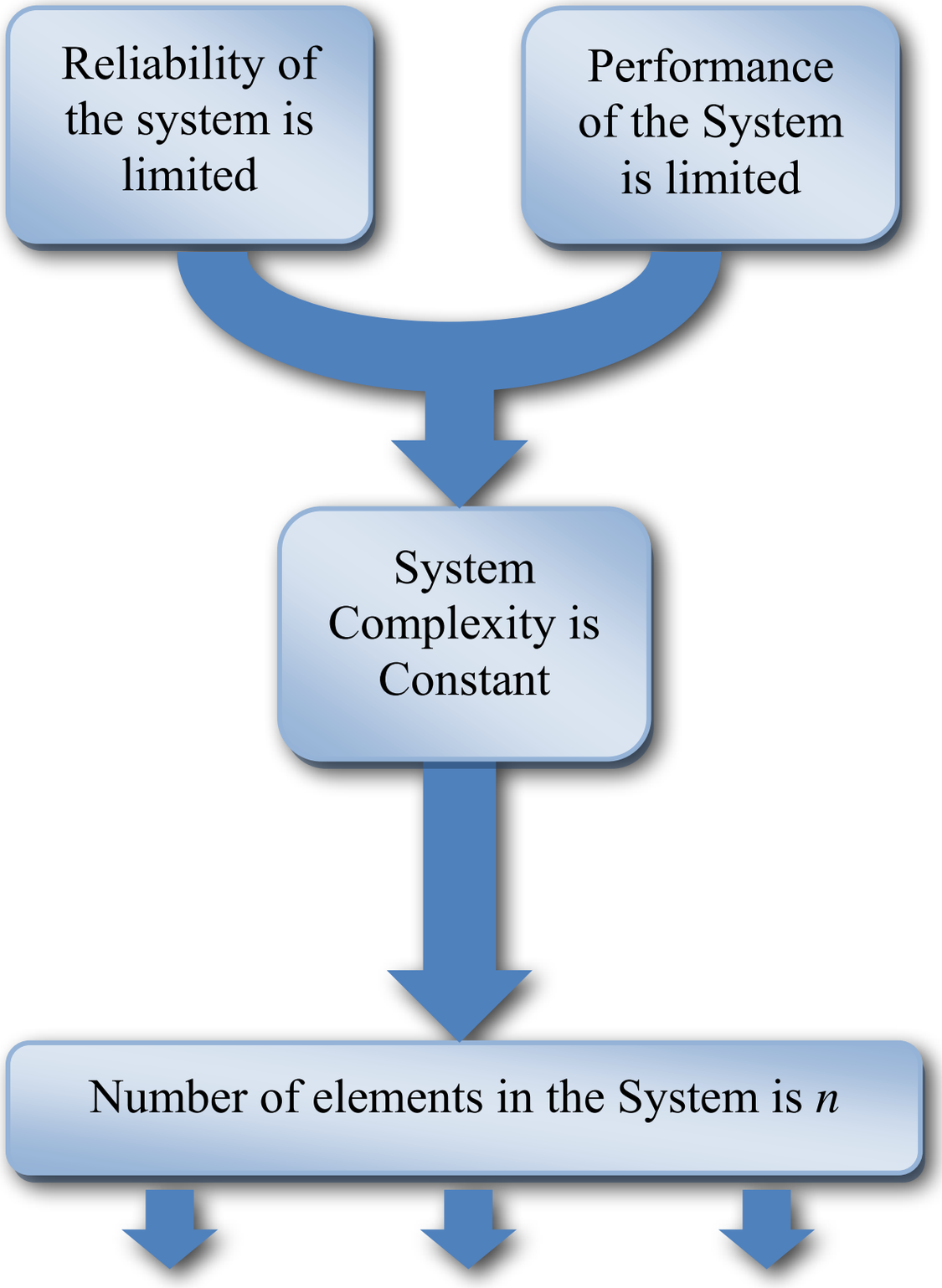
- Health monitoring - tests and checks
- Recovery point support
- Recovery point handling (organization, HW use)
- Recovery point search
- GAFT support: reconfiguration control - a syndrome

System Software & Hardware for future: PRESSA

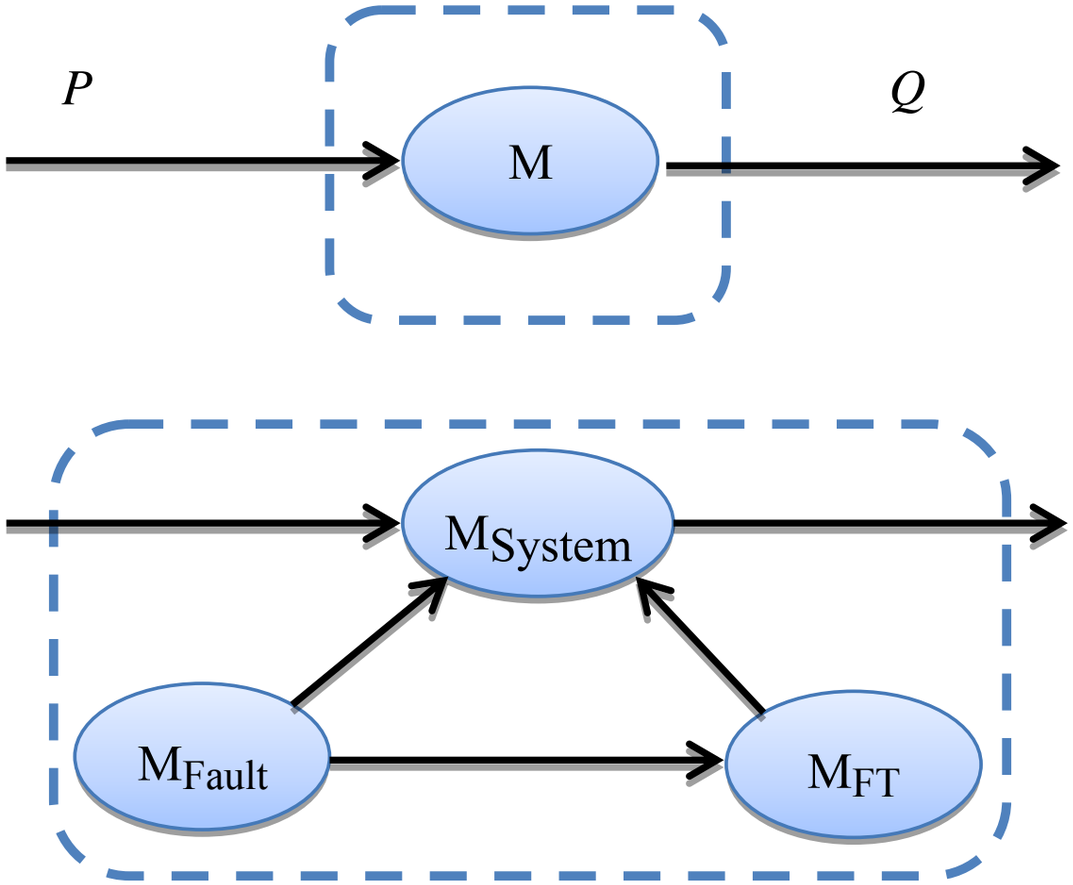
R&D principles for computer systems



Reliability vs. performance in computer systems



Model of fault tolerance: introduction of GAFT



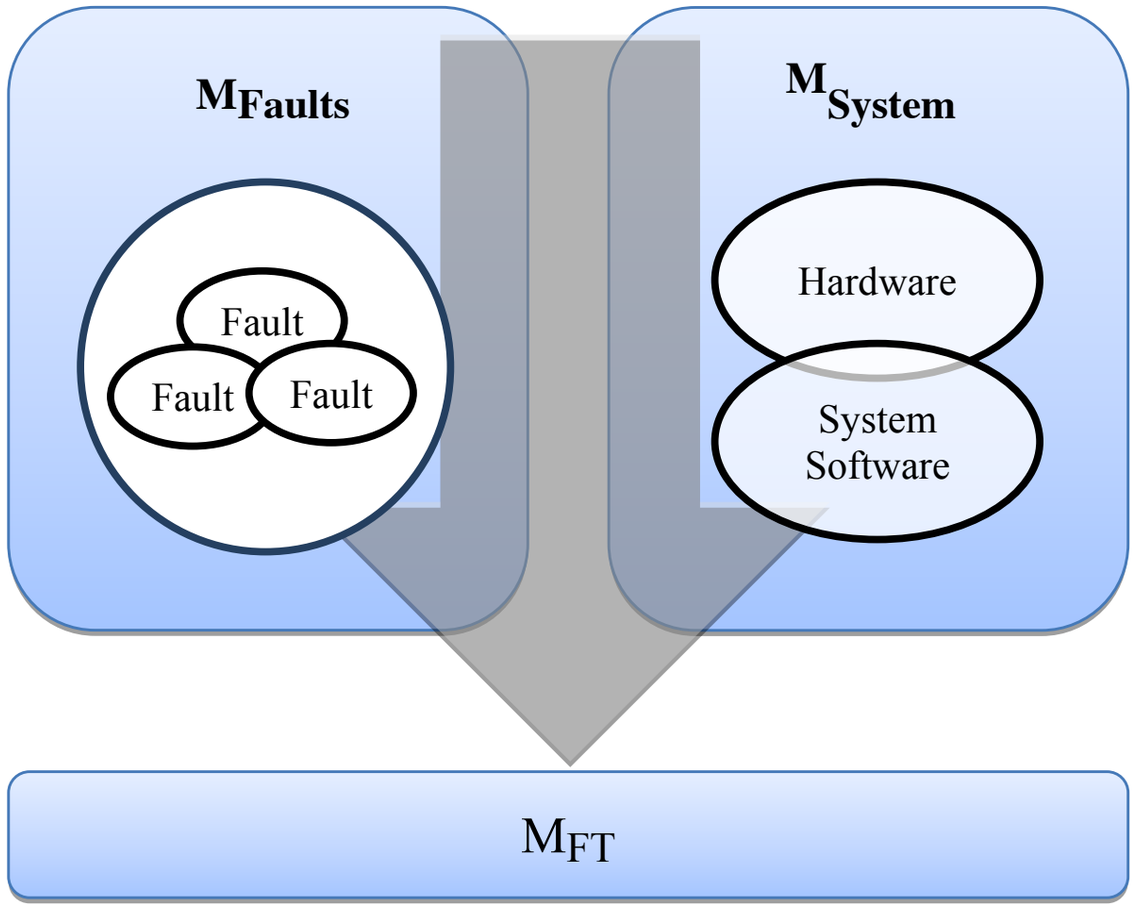
FT property will be achieved if...

Generalized Algorithm of Fault Tolerance (GAFT)

- Detecting faults
- Identifying faults
- Identifying faulty component
- Hardware reconfiguration to achieve a fault-free state
- Recovery from correct state(s) for both: the system and user software

Model of fault tolerance GAFT and HW & SSW

Hardware and System Software for Fault Tolerance



GAFT in HW and SSW

```
IF error is detected THEN

  Determine the fault type;
  IF the fault is permanent THEN
     Locate the faulty component;
     Reconfigure the HW by
     excluding faulty unit;
  END;

  IF the fault affected the SW THEN
     Locate faulty program states
        and find the correct ones
        to continue;
     Recover the system from
     preliminary stored correct SW
     states
    END;
END;
```

New property achieved

GAFT implementations using redundancy

Step	*Description*	*Redundancy Types*					
		HW(I)	HW(S)	HW(T)	SW(I)	SW(S)	SW(T)
0	PERIODICALLY DO Create recovery point END		7			7	
A	IF error is detected THEN	2	1,3,9	1,2	8	6,9	6
	ELSE						
B	Determine the fault type	2	1,3,9	1	8	6,9	6
C	IF fault is permanent THEN						
D	Locate Faulty Element	2	1,3	1	8		
E	Reconfigure Hardware		10			10	
	END						
F	IF hardware has been reconfigured OR software is affected		3		8	6	5,6
G	Locate faulty software states					7	7
H	Recover software		9			7,9	7
I	IF hardware has been reconfigured THEN						
J	Reconfigure software		10			10	
	END						
	END						
K	CONTINUE						

GAFT might be implemented (and fault tolerance achieved) using redundancies of:

Information (I)
Time (T) or
Structure (S)

this way we **think..**

Implemented *in* and *by*:

Hardware (HW) and/or
Software (SW)

this way we **do**

Nr.	*Name*	*Redundancy type*	*Description*
1	HW checking	$HW(\delta S, \delta T)$	Each hardware component such as processor, memory, controllers has built-in checking schemes to detect faults.
2	processor instruction re-execution	$HW(\delta I, \delta T)$	The processor itself has measures to detect faults during execution and can abort and restart the currently running instruction
3	Triplicated memory	$HW(3S)$	The memory chips are triplicated and a voter compares the output of the three memory chips. If a deviation is detected, the majority voting is used to identify the faulty chip and the faulty value is rewritten. Read after write ensures the proper storing of the data.
4	Duplicated storage device	$HW(2S)$	Storage devices such as flash cards or hard disks are duplicated. Note that this feature does only provide fault detection but not recovery
5	Duplicated program run & input validation	$SW(2T)$	The same program is run twice with the same input data set. The output of both programs is then compared. Prior to running the program, the input data is validated to conform to a certain pattern and range.
6	Checkpoints	$SW(\delta T, \delta S)$	Periodically executed checking functions for checking software and hardware, implemented in pure software
7	Recovery points	$SW(\delta T, \delta I)$	Recovery points are points in time when the complete system state is consistently stored on a permanent storage device such as flash discs or hard disks. They are either triggered by software or an external interrupt.
8	CRC	$SW(I)$	The data stored to the external storage device is protected by a CRC-32. This allows the identification of incorrect data but no recovery.
9	Watchdog	$HW(S)$	As an ultimate resort, a watchdog is used to restart parts of, or the whole system. Hardware based watchdogs can typically on restart the whole system at once.
10	Reconfiguration facilities	$SW(S)$, $HW(S_1, S_2)$	If the hardware failed, the software can reconfigure the hardware to exclude faulty components. In addition, the software can start alternative software version which need less resources to adapt to the new hardware configuration.

GAFT vs. ontologies... personal comment

Now in BOLD: Classification of system redundancy in terms of

Information (I), Time (T), Structure (S)

can be used to implement Fault Tolerance (GAFT).

Fault tolerance, as a process, has to be implemented through hardware and system software combination. Both: concept and implementation form a theory that allows to *analyze*, *control* and *predict* behavior of the system with new properties.

An example of GAFT extension: "Method and apparatus of system safety" (patent http://it-acs.co.uk/files/GB2448351B.pdf)

In contrast to popular wave of various kind of ontologies that execute *descriptive function* of knowledge - GAFT and rigorous classification of redundancy analyze & predict system behavior.

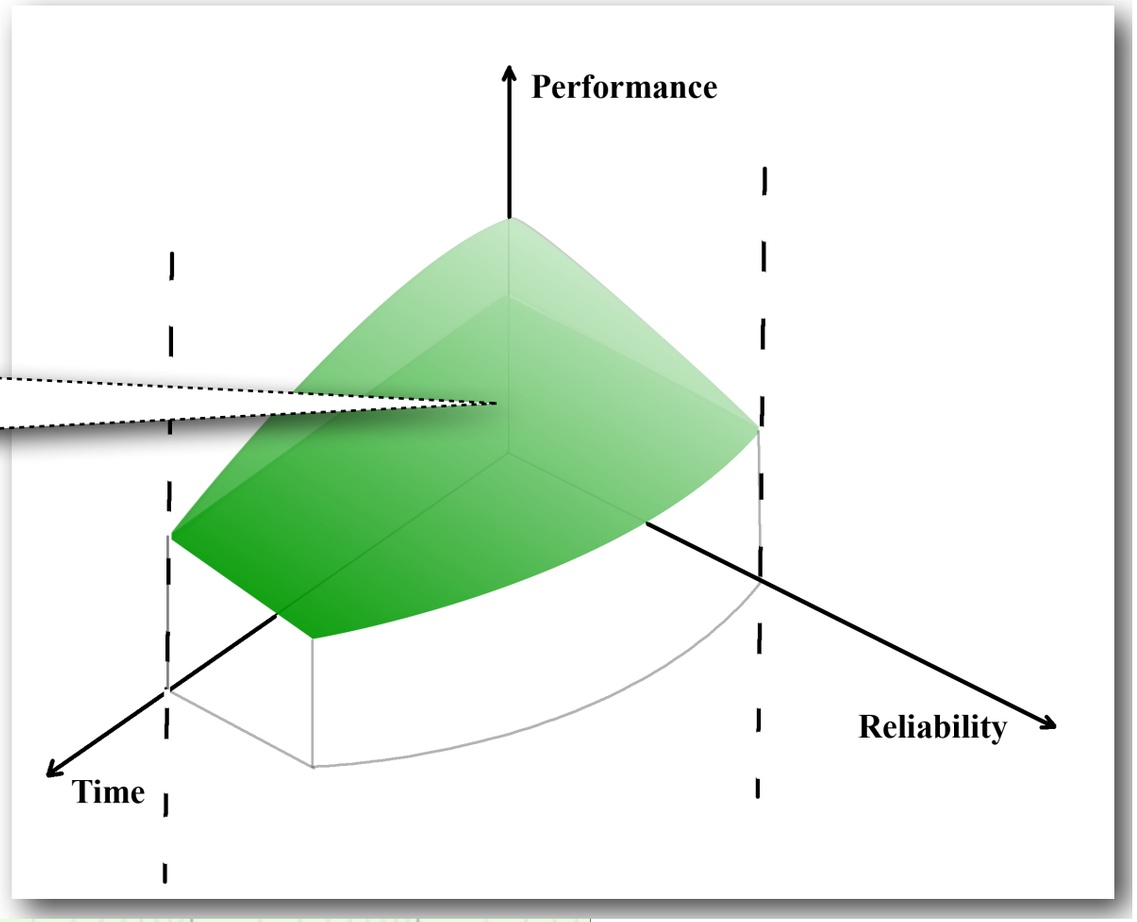
GAFT & redundancy theory vs. ontologies

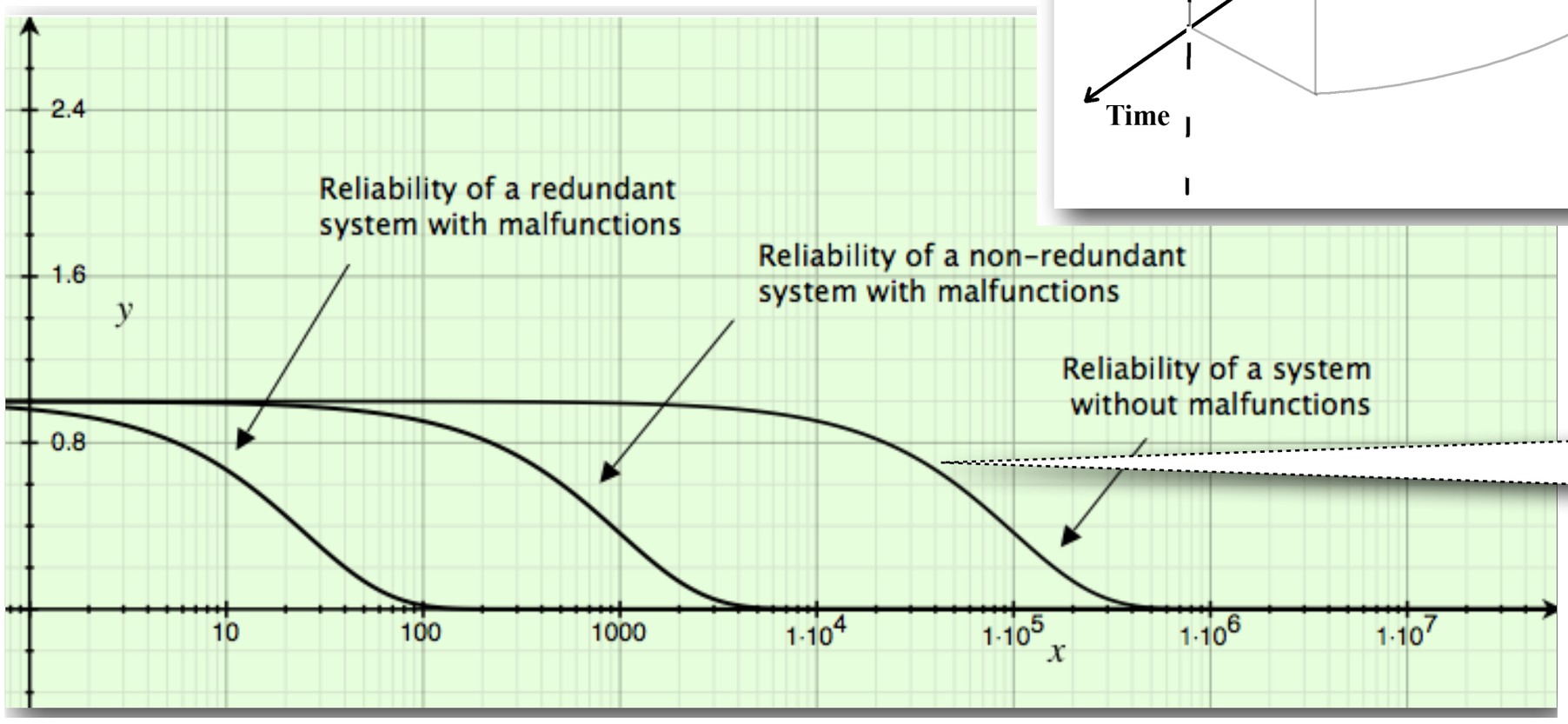
	Definitive function (DF)	*Characteristic Function (CF)*	*Predictive function (PF)*
GAFT	+	+	+
Ontologies	+	?+	

Power of any theory is in predictions and our ability to use them

GAFT impact on performance and reliability

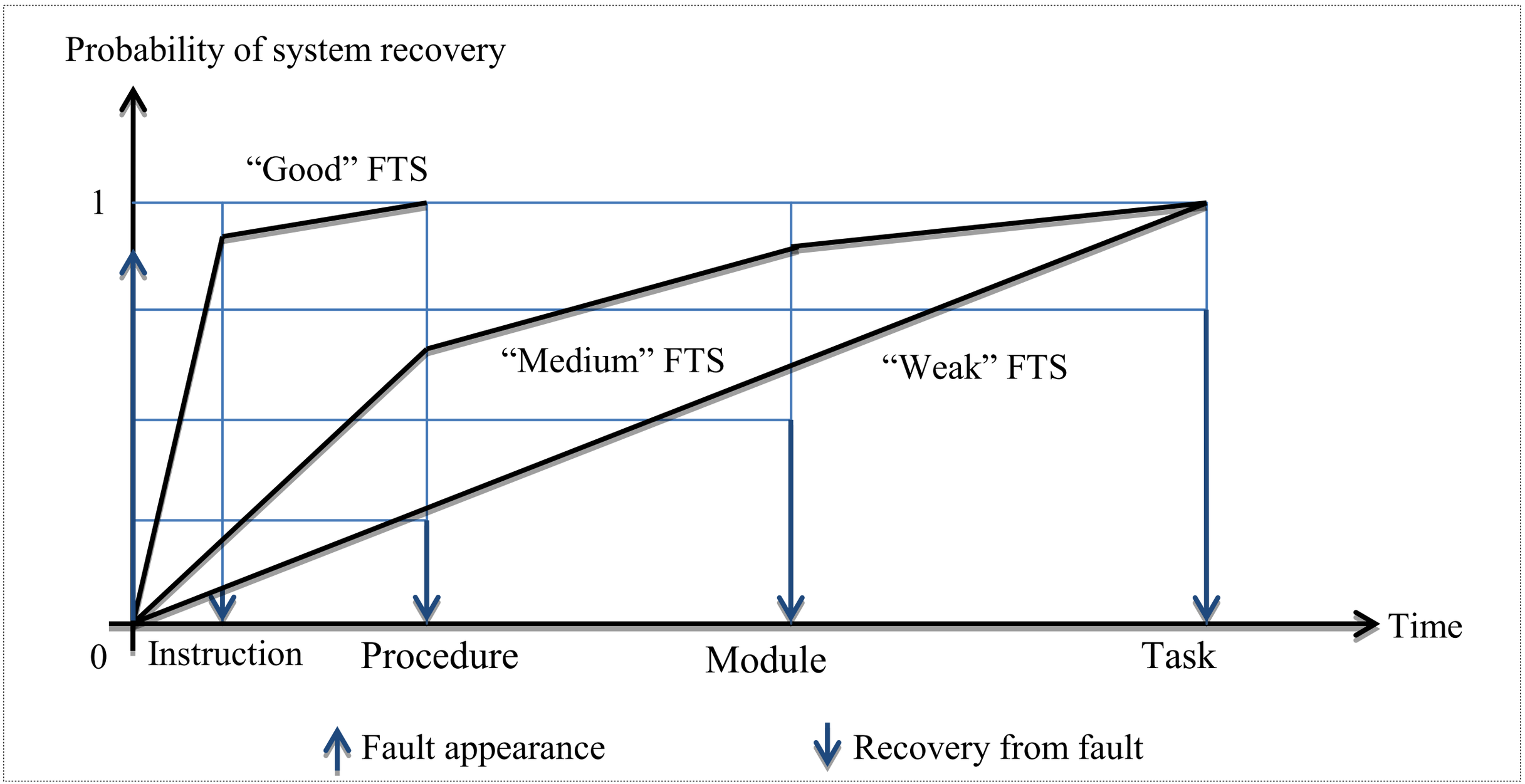
Performance & Reliability of our systems should be within required zone for the whole operation cycle...





Reliability is achievable with system software support

GAFT implementation using SSW soultions



NB: if probability of recovery ≠ 1 the system is **NOT** fault tolerant !!!

Language reflection of FT systems

New Features

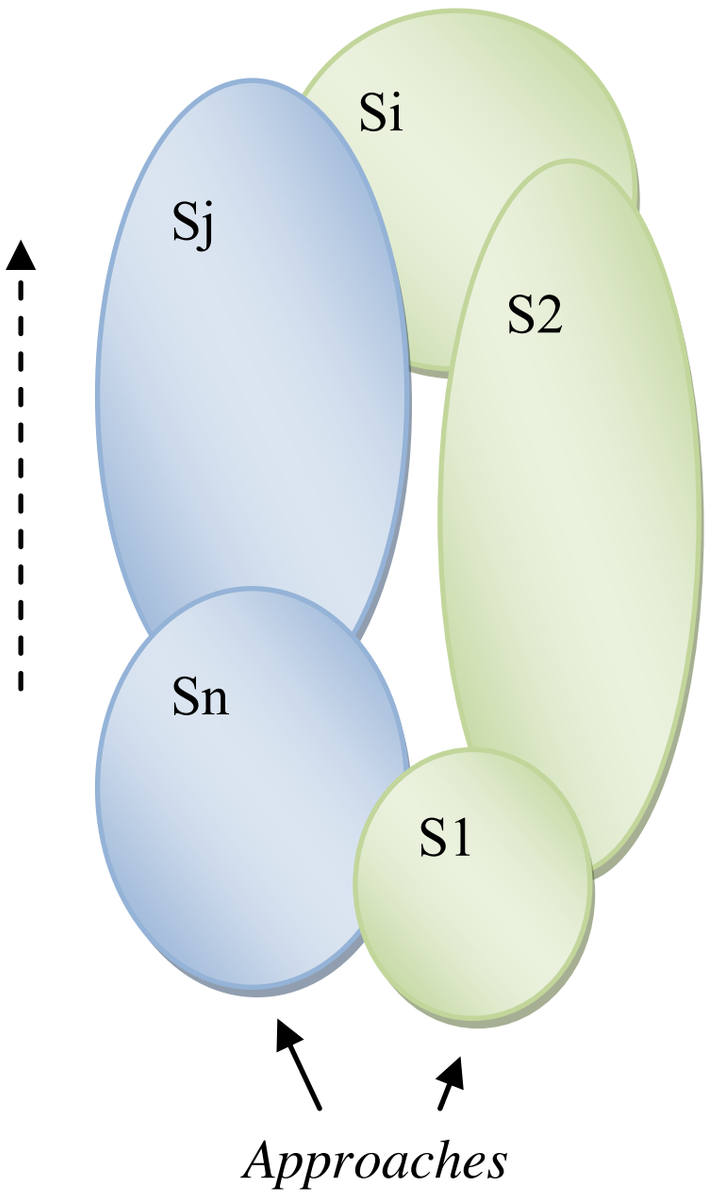
Real Time

HW (Timers, RISC structure of processor etc

Language (Limitation of the language constructions that complicate RT capability)

OS (Management of timers and task scheduling with RT constraints

AP (Application specific schemes of RT)



Fault tolerance

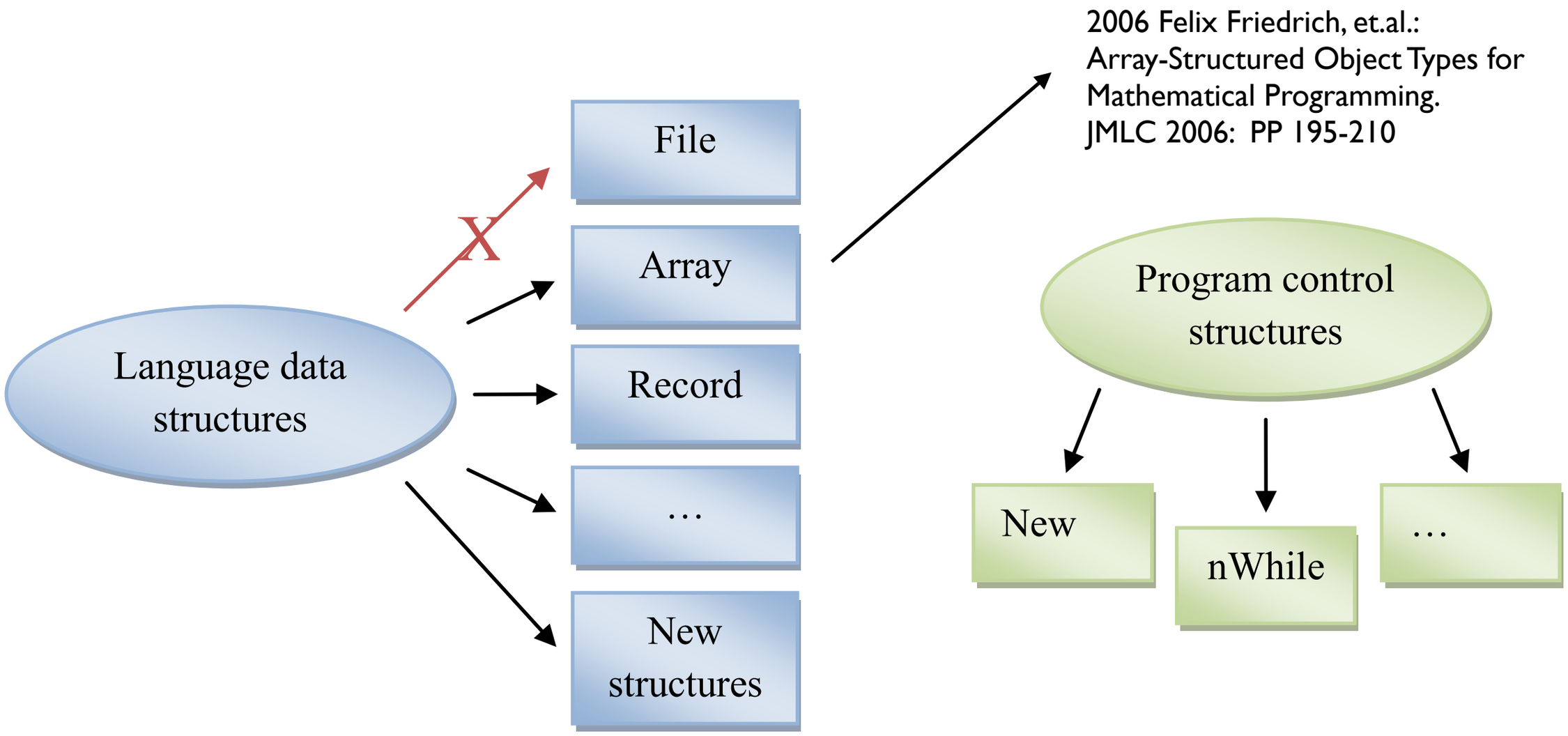
HW (Majority schemes, Hamming codes)

Language (check points, recovery points, at language level)

OS (management of check points, recovery points, synchronization, HW reconfiguration)

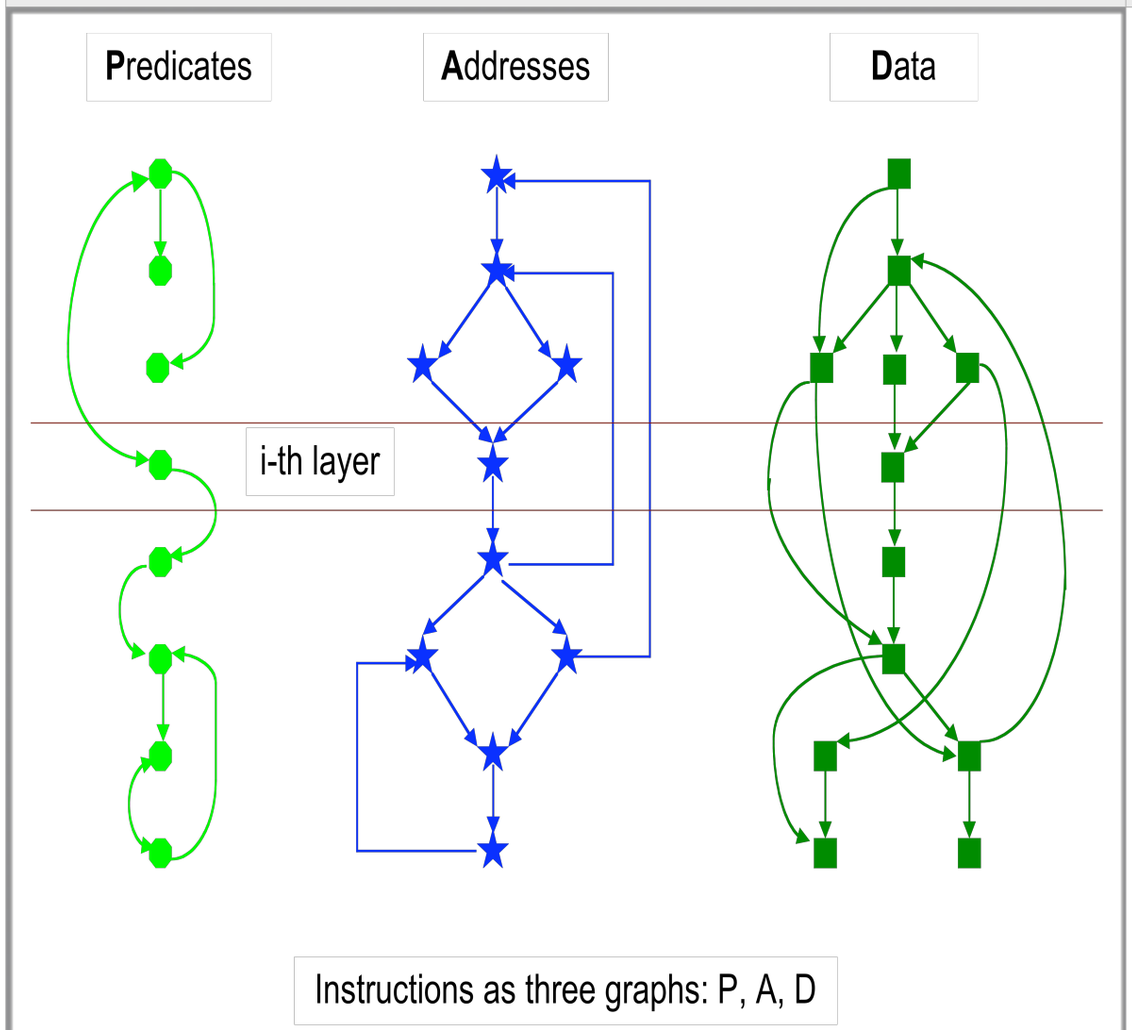
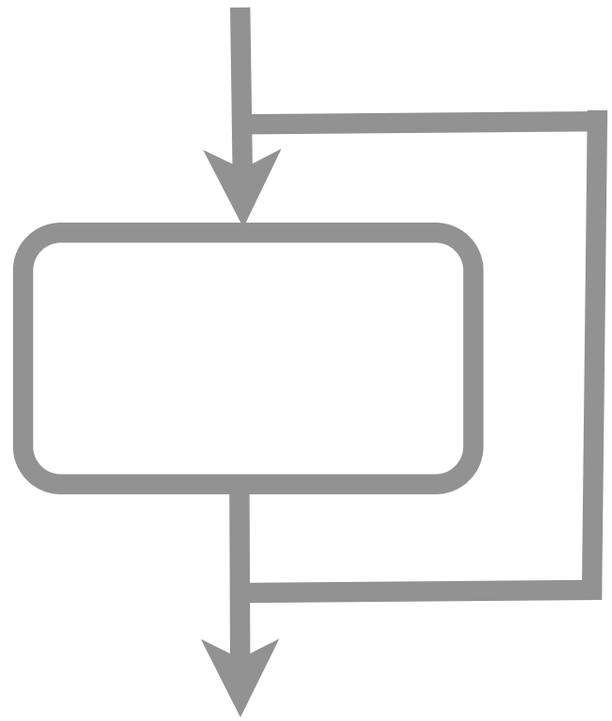
AP specific realization of possible hardware deficiency solutions

Data structures and control operators 4 FT



File structure must be modified, Control operators - upgraded

State of the systems, Control operators

C,D,P model	Loop
	

Level ***i*** is what has been changed in state of hardware.

All: control, data and conditions involved must be preserved (to be able to recover...)

Control using nWhile

So far there is no clear separation of the actions to react on exceptions at the language level for operators of repetitions.

This is because awaiting of an event might be perfectly valid action or ...

endless wasting due to hardware fault that has happened.

New **nwhile*** loop can be useful

Takaoka* suggested new control operator, actually without thinking about embedded system issues...

This operator was called **nwhile**, and looks like below:

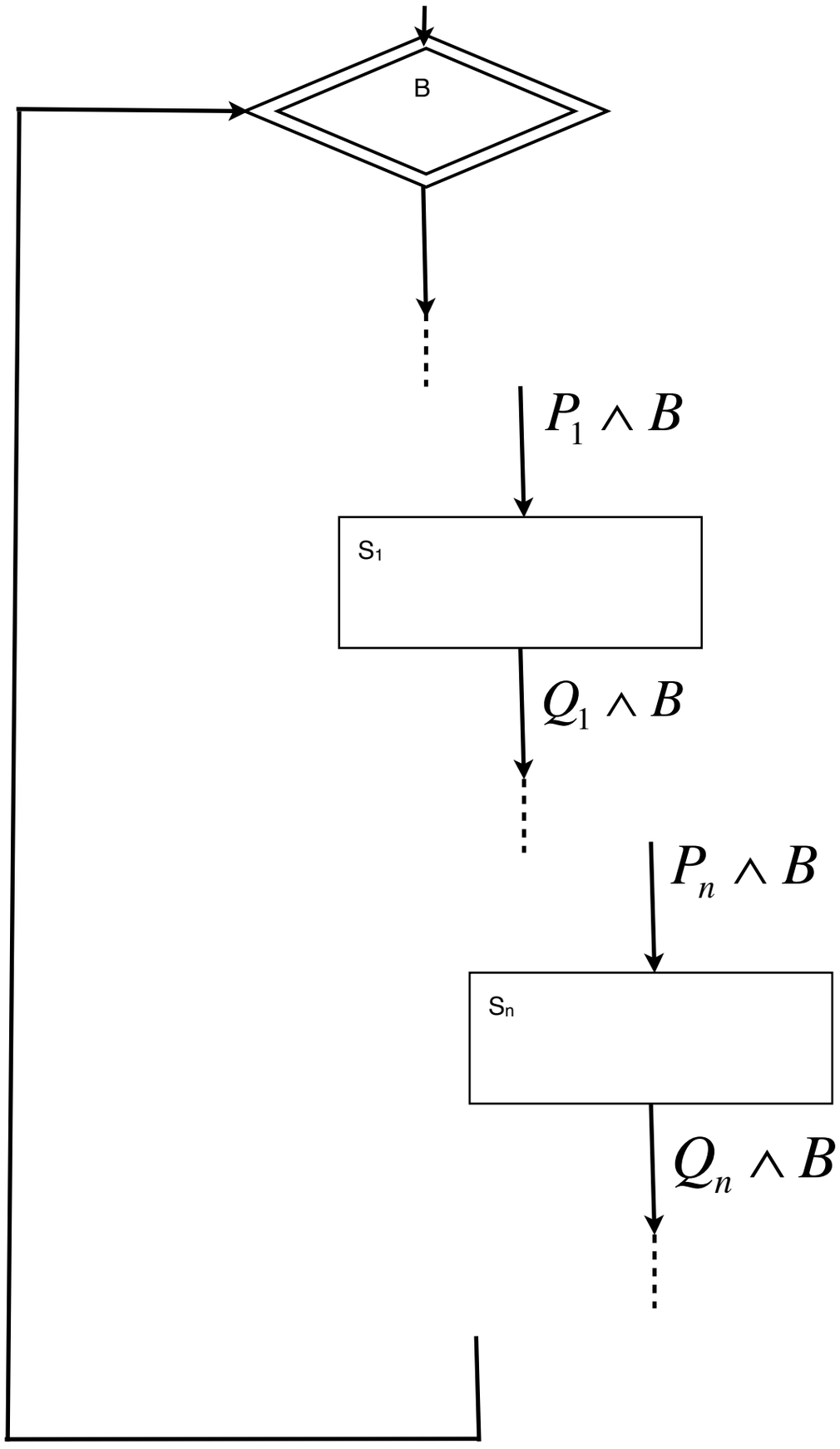
nwhile *B* **do** *S*

where B is condition to enter the loop and S is body of the loop.

Introducing precondition P and post condition Q for this structure Takaoka suggested to use several assignment statements S_1, S_2, S_3,... S_N in S which affect the condition B and therefore P_1 P_2, P_3, P_4,... P_N that held immediately before S_1, S_2, S_3,... S_N under precondition P.

* New While loop [*The semantics of new while loop, Tadao Takaoka, The Computer Journal, Vol.29,No 1, 1986*].

Control using nWhile



Then we have an inference rule:

$$\frac{\prod_{i=1}^{n}\{P_i \wedge B\}S_i\{Q_i\}, \prod_{i=1}^{n}(\neg B \wedge Q_i \supset Q), P \wedge \neg B \supset Q}{\{P\}nwhileBdoS\{Q\}}$$

What it gives us? Actually, a lot.

Writing a program we use loop operators as usual, but during compilation System Configurator should introduce other S_2-S_n conditions for loop exit that might be connected with computer state changes including hardware faults, timer run out or other interruptions, including interaction with other processes.

Then in case of hangs of the loop due to problem within hardware and/or arrival of another signal we are able to break loop execution and make it visible...

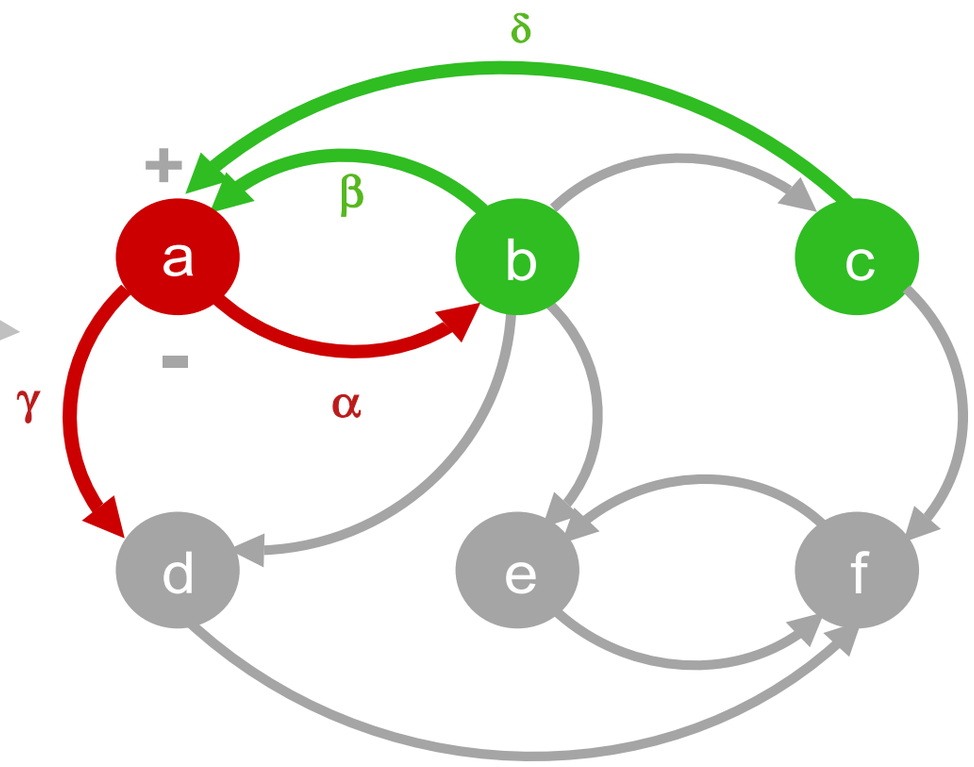
hardware state change is reflected immediately *within* program control construction and ... we are not using brutal force of waiting or waste of vast amount of another redundancy... still being uncertain...

More control: Fault Tolerant Semaphores

Concurrency and parallelism confusion;

What we start in parallel eventually will end up with concurrency... and (possibly) vice verso...

Resolving confusion - Graph Logic Model (GLM)



XOR and AND will resolve confusion. Known solutions *semaphores*, *monitors* are all down to XOR.... But we should be using both operators...

$$a : XOR_{-}(\alpha b, \gamma d), AND_{+}(\beta b, \delta c)$$

$$a : XOR_{-}(\alpha b, \gamma d), XOR_{+}(\beta b, \delta c)$$

Still... Why Fault Tolerance is required?

...Waiting mad driver to release train handle might be costly... (*Jethro Tull, Locomotive breath*)

Time redundancy - waiting for few *ms* for processes duration within dozens of *ns? ... It* is NOT *the solution*.
Information and structure redundancies should be used instead...

GLM - is structural redundancy... Information?

...old Charlie stole the handle and the train it won't stop going no way to slow down...

Control: Fault Tolerant Semaphores -11

Mad driver, also known as “dining philosopher” should be trained to be polite...

We need to teach our dining philosophers to be polite, and ... "die like a man", especially when they are in critical section.

What does it mean? The one, being in critical section, if sick or dies must:

- Return all spaghetti, forks - resources he uses
- Inform the rest by "I am dying" message...

Then “the rest” (Runtime system in our case...) has to :

- Reduce number of voters for further voting in sessions of concurrency resolution
- Mark a messenger as suspected (not excluded, or dead) and place in a special pool
- Schedule a “reincarnation procedure”

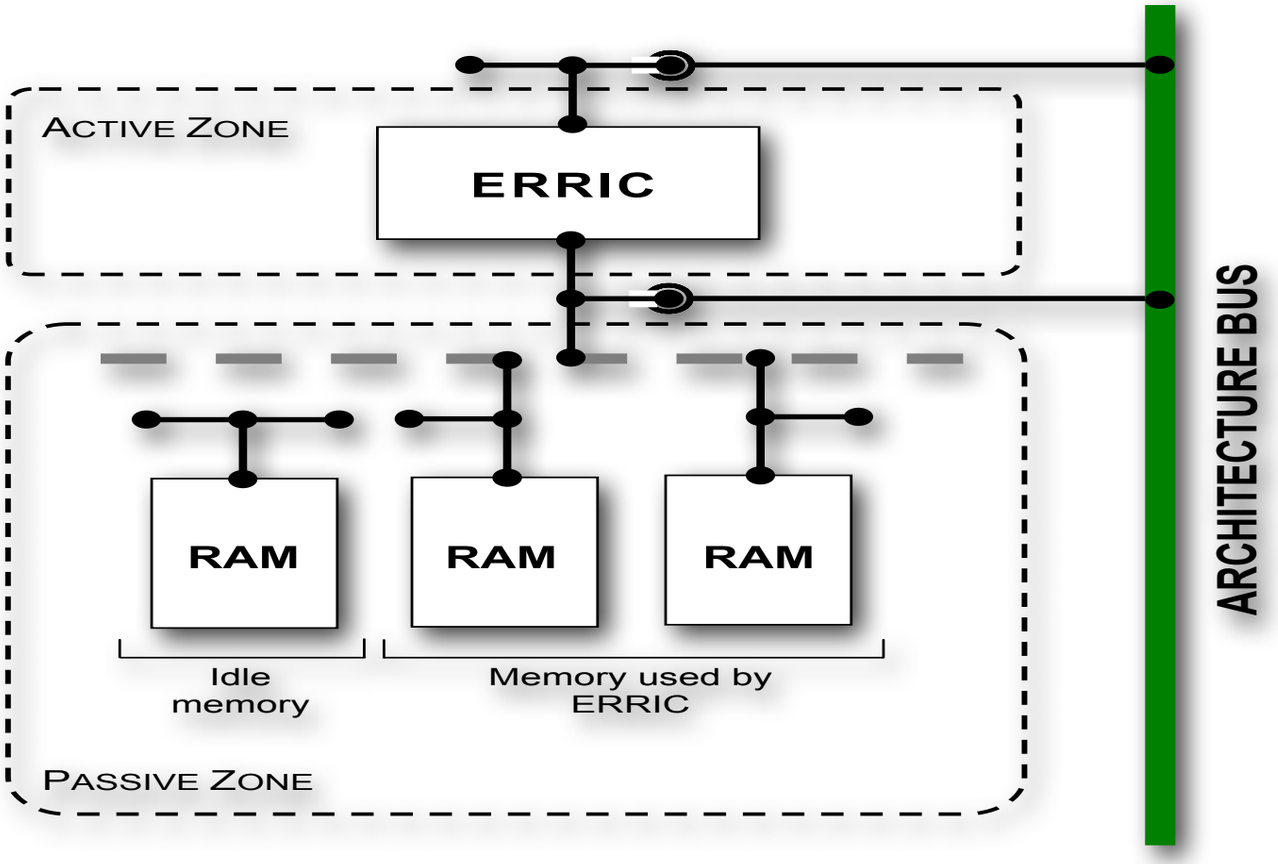
(...power of malfunction might be big, duration long (200ms+), but “treatable”...)

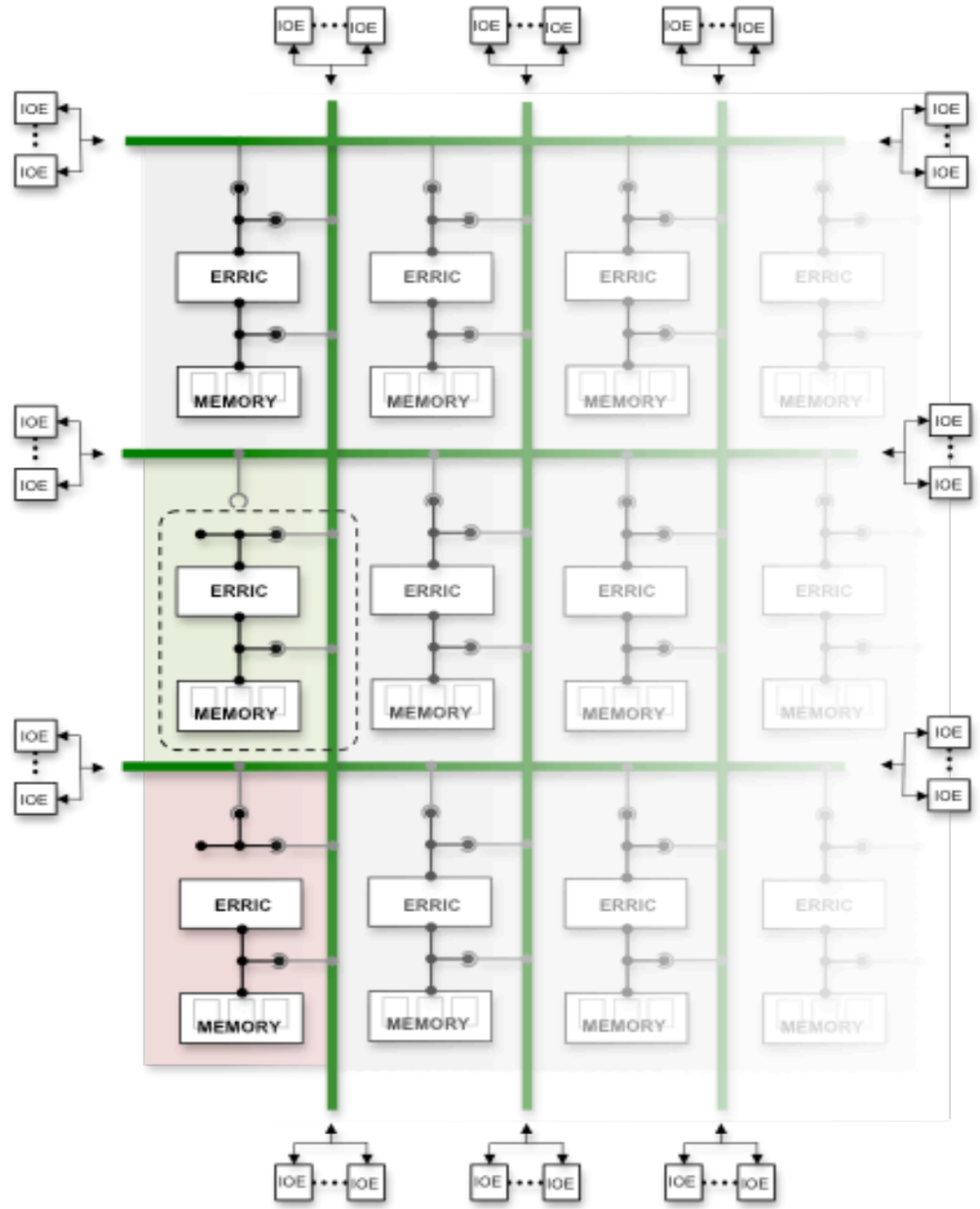
This scheme has been called FT semaphores...

FT Semaphores hardware support: T-logic scheme

for one computer

for multiprocessor system



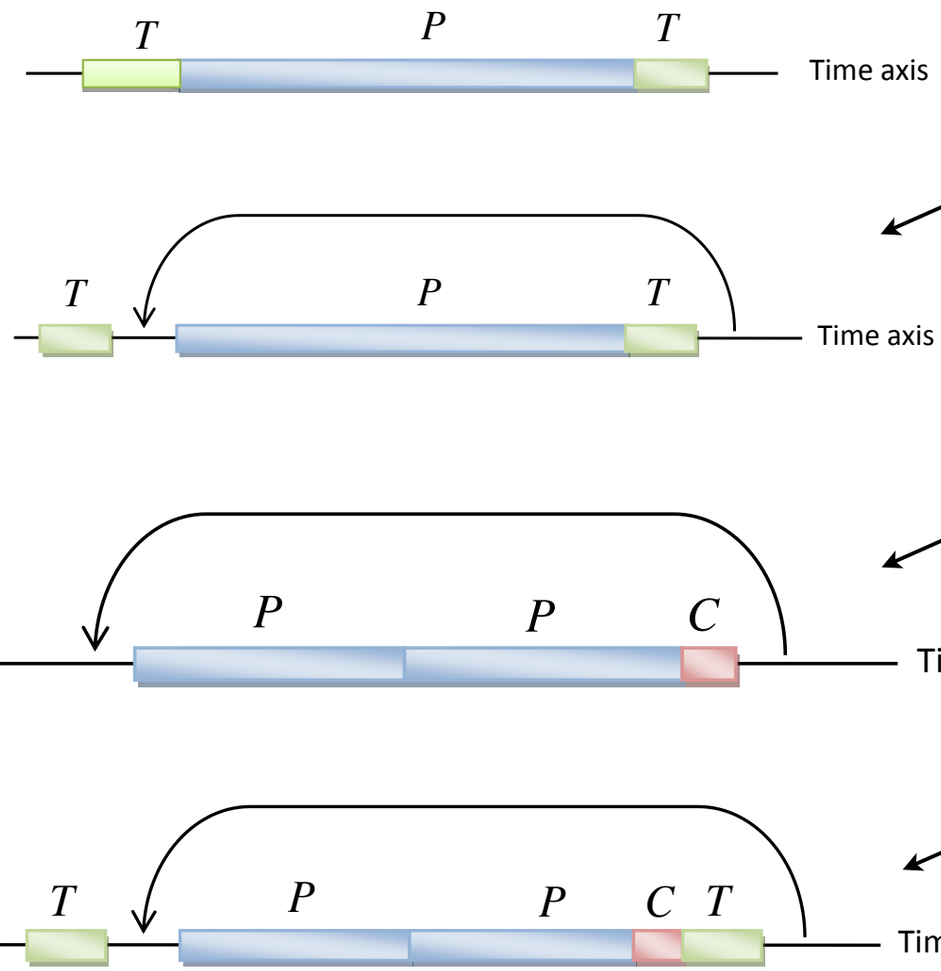


- HW element "suspected" should "switch itself" - (left RAM above);

- System should be able to return it in action after full-size check, if it was recovered.

GAFT: System checking by system software - I

all about testing at the level of program -



Regular sequence of program with test of hardware integrity to detect permanent faults

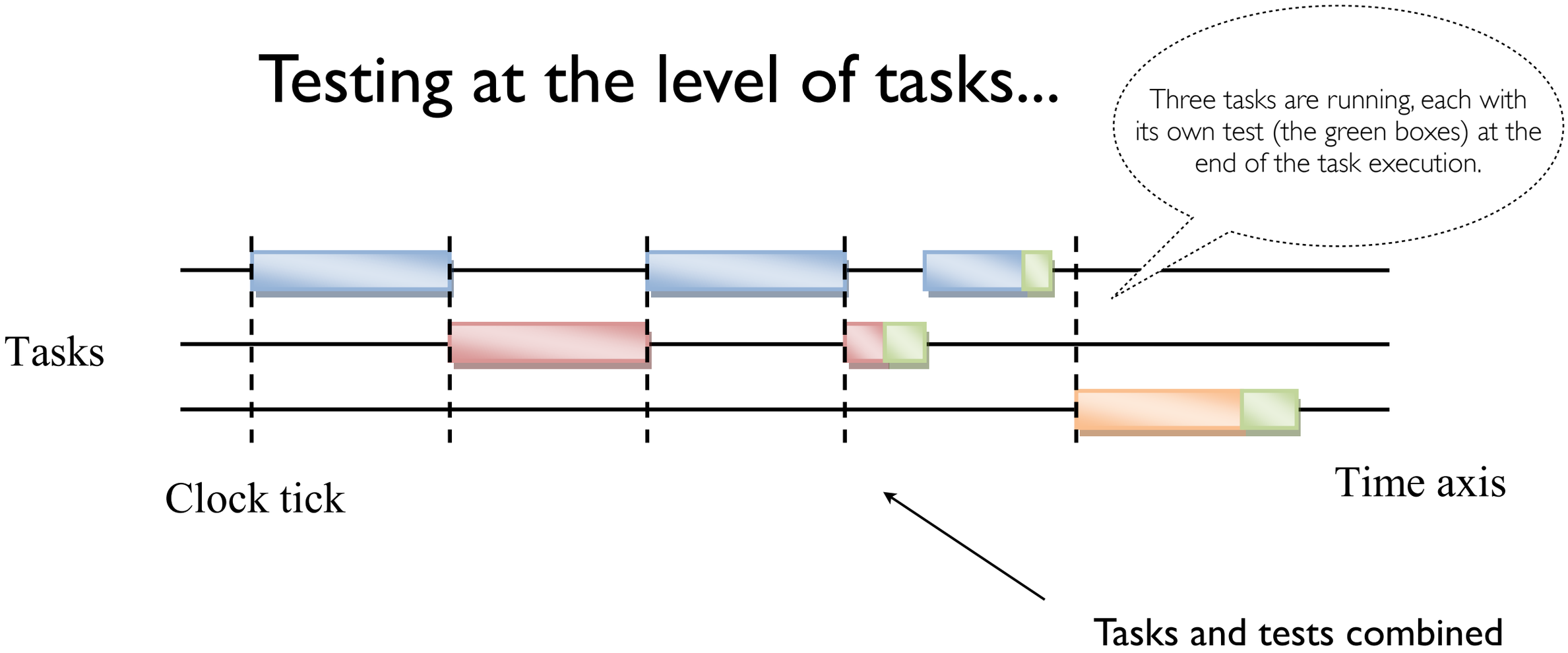
Regular sequence of program to detect hardware malfunctions

Regular sequence of program with test of hardware integrity to detect permanent faults and malfunctions

A testing phase is required initially at boot up time to guarantee the correctness of the hardware; periodic test is required before and after the execution of a program. The applied tests might vary in depth (coverage), type of faults and the set of the tested hardware.

GAFT: System checking by system software - II

Testing at the level of tasks...

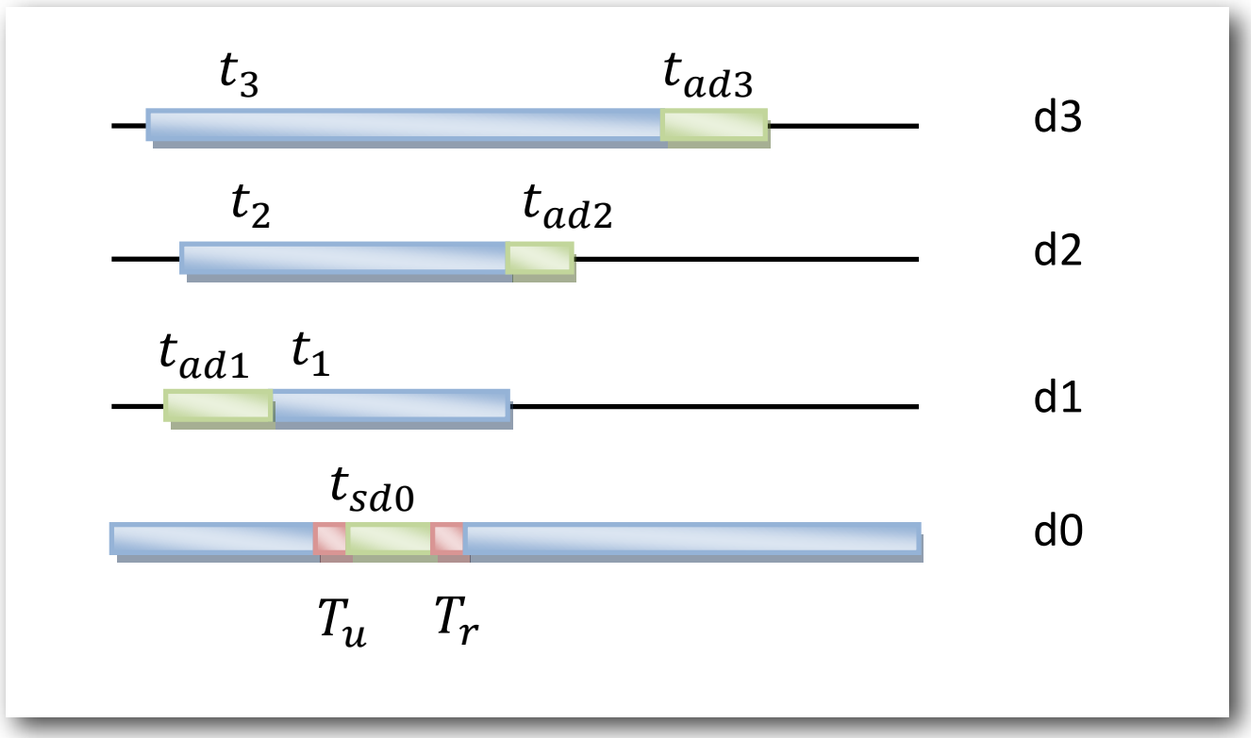


NB1 *If condition of hardware component Ui has implicit dependencies on component Uj, test of Uj must be executed first.*

NB2 *It is wise to wait task completion and then run test of hardware instead of stop, unload and reload task after test.*

GAFT: System checking by system software - II

Testing at the level of tasks...



Tasks are running, each with its own test (the green boxes) at the end of the task execution and, when required unload and reload tasks

An algorithm for scheduling and imbedding tests of hardware used by each task should suit various number of tasks and time constrains for group of task completion...

*The test of task **i** is performed asynchronously, if it is possible to schedule it in the timeframe **ti** to **di** as long as all other tasks can still meet their deadlines and only one test is executed simultaneously. Otherwise, execute the test synchronously. More see pp 68 -79 (http://www.it-acs.co.uk/book.html)*

GAFT: Recovery by System Software

Language support

Run-time system support

Recovery points (RP)

RP formation

RP formation rules

- **Time**
  - Timer of and for:
    - Process
    - Module
    - Task
    - System RQ
    - Runtime control
- **Structure**
  - Plain: back-up for
    - Task
    - Program
    - Full size back-up
  - Structure-wise:
    - Hierarchical (static)
    - Save as you go (dynamic)
- **Information**
  - Condition of:
    - a System:
    - a Task (dynamic priority)
  - Run-time control
  - Check sums of RPs

RP monitoring/handling

RP structure support

- "Fake" formation
- Check-sums
- Indexing
- RP handling
- Hardware support

Search of correct RPs

- Linear
- Binary
- Modified linear
- Power of search
- Search control

RP formation: structure-wise scheme

Static, Language

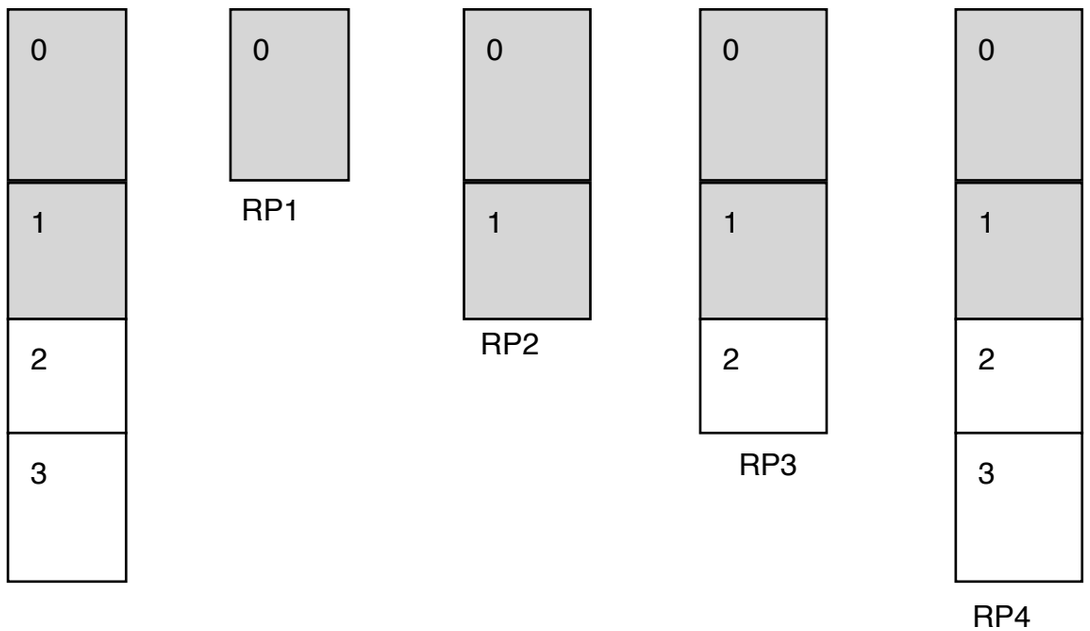
Recovery points (RP)

Language with Run-time system support

Hierarchy of program

N Wirth's structural programming can be exploited:

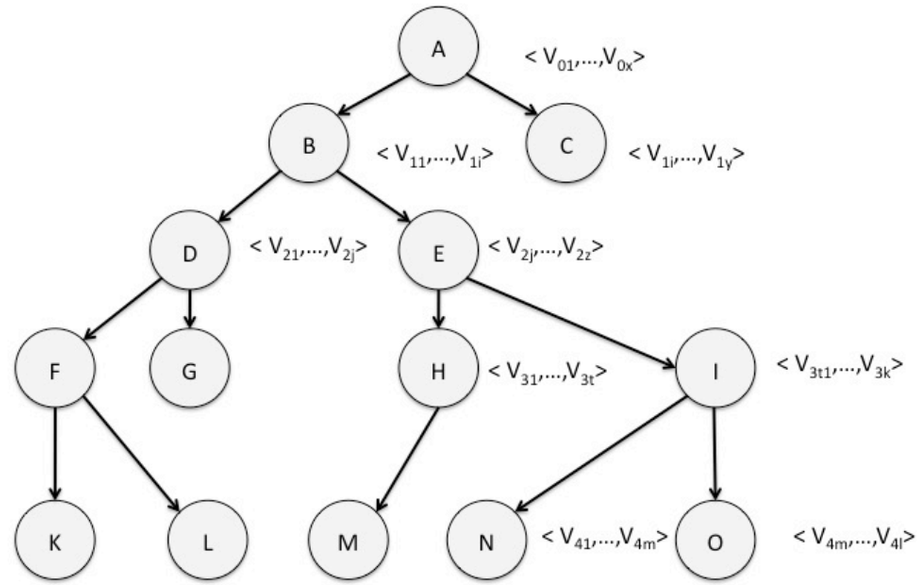
1) Structural features and limitation of visibility for lower layer variables reduce a volume of recovery points;
2) Only variables that are accessible at the level are required to save at recovery point;



Save as you go

Along the tree, selected path:

1) Select subset of visible variables you use
2) Create a "Key" , i.e. collection of variables to a given leaf



$K=<V_{01}...,V_{0x}><V_{11},...,V_{1i}><V_{21},...,V_{2z}><V_{31},...,V_{3k}><V_{41},...,V_{4m}>$

Recovery Points Support Summary

- The original program code should be re-processed, introducing a generation of recovery point at the beginning of each hierarchy level;

- During compilation a data structure with a list of accessible variables should be formed for each level of a program nesting;

- The program begins an execution of each successive level by calling run-time system indicating the level number, the number of accessible modules and list of variables;

- Run-time system has to monitor keys along executing a program and generate checksums along execution of recovery point;

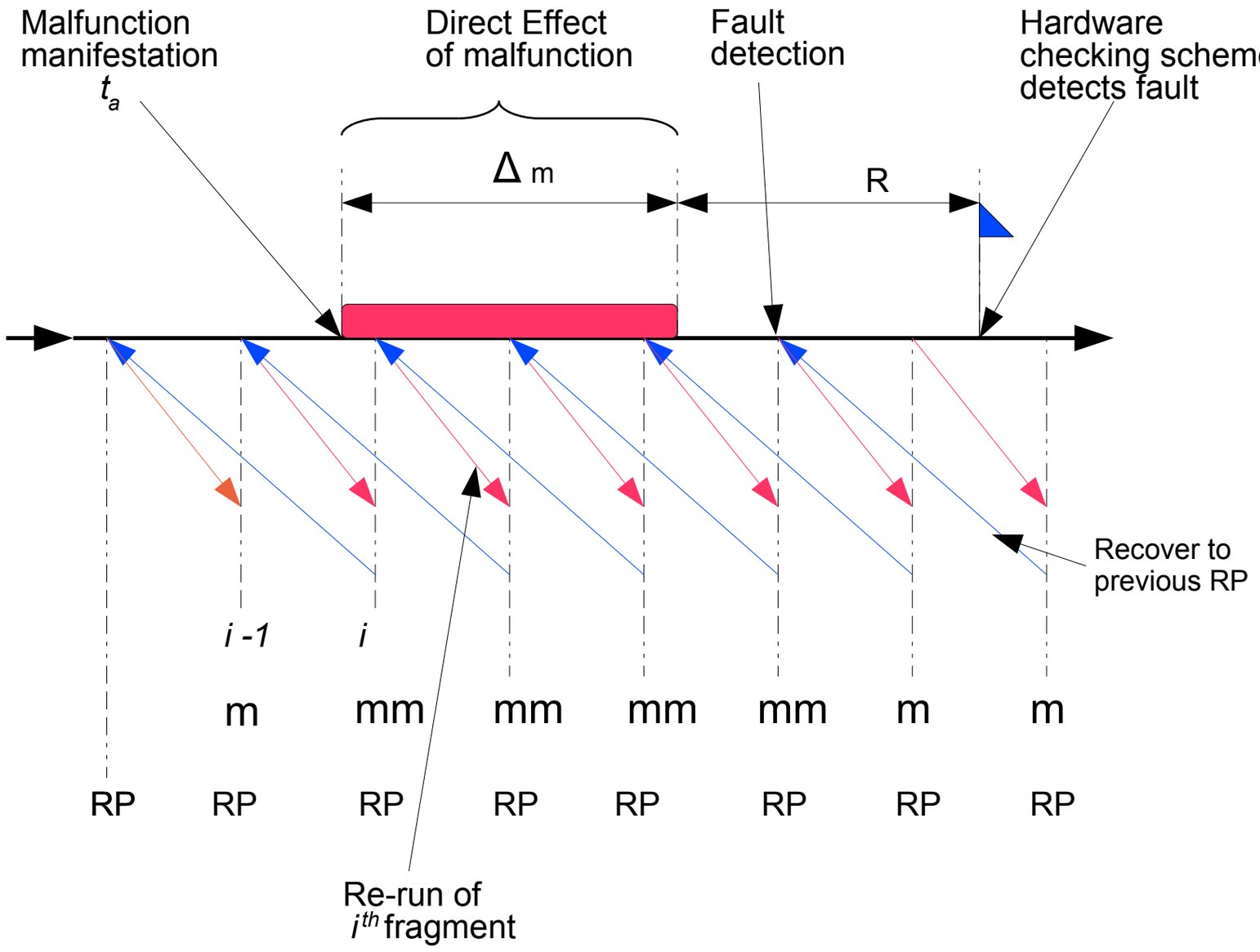
- Run-time system can "simulate" recovery point formation when it is required;

- Execution of "generate recovery point" action consists of recording of variables accordingly key generated scheme along the execution a program: *Save as you go*

NB. Efficiency of "static and dynamic economy" methods of recovery point formation was proven to be at the order of magnitude better than other known schemes of recovery point schemes.

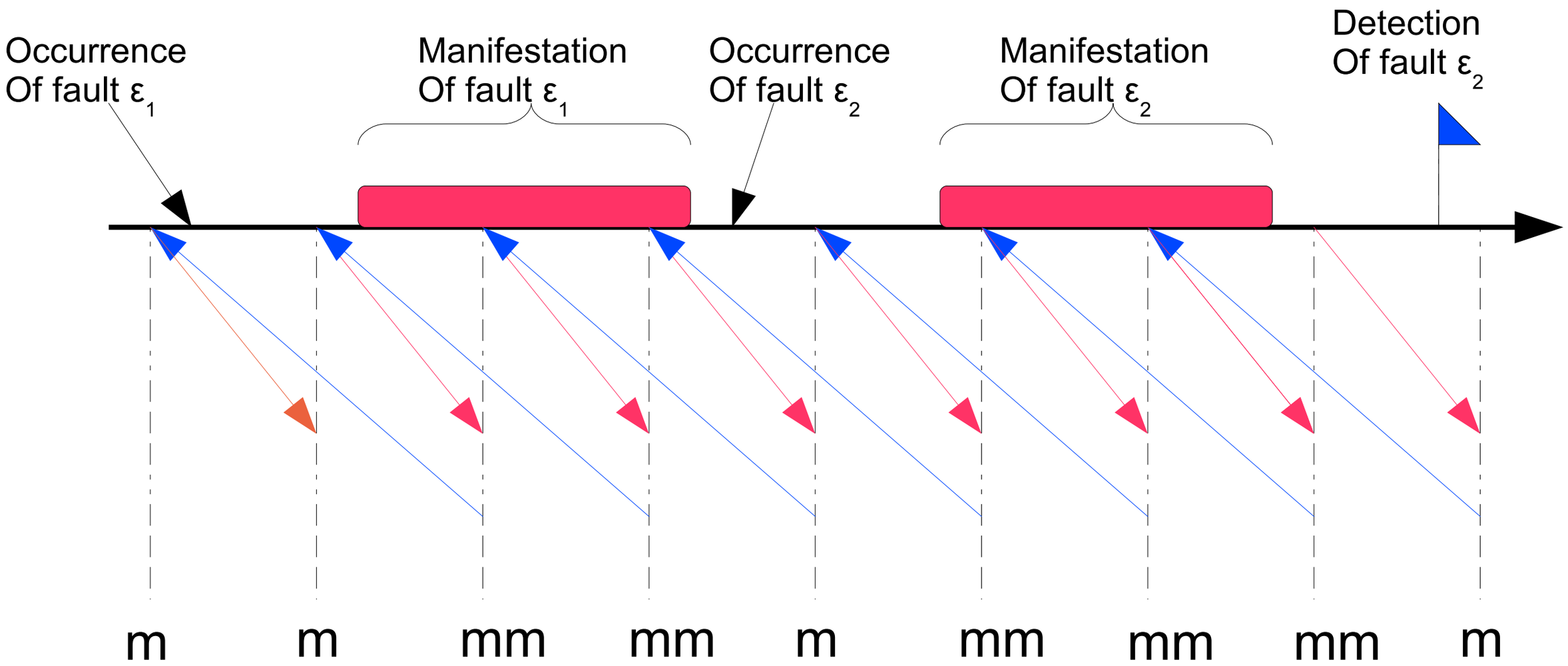
Searching of correct state of a program: MLR

...Faults might stay latent in the system for a long time until they trigger an error, i.e. it implies that the last recovery points have been damaged by the latent fault...



Modified Linear Recovery Algorithm (MLR)

Searching: MLR powers multiple faults



During search MLR creates Check Sums (CS) and compares them with previously created CSs of RPs - sequence of matches and mismatches *m* and *mm defines MLR termination rule*.

Recovery support : Hardware

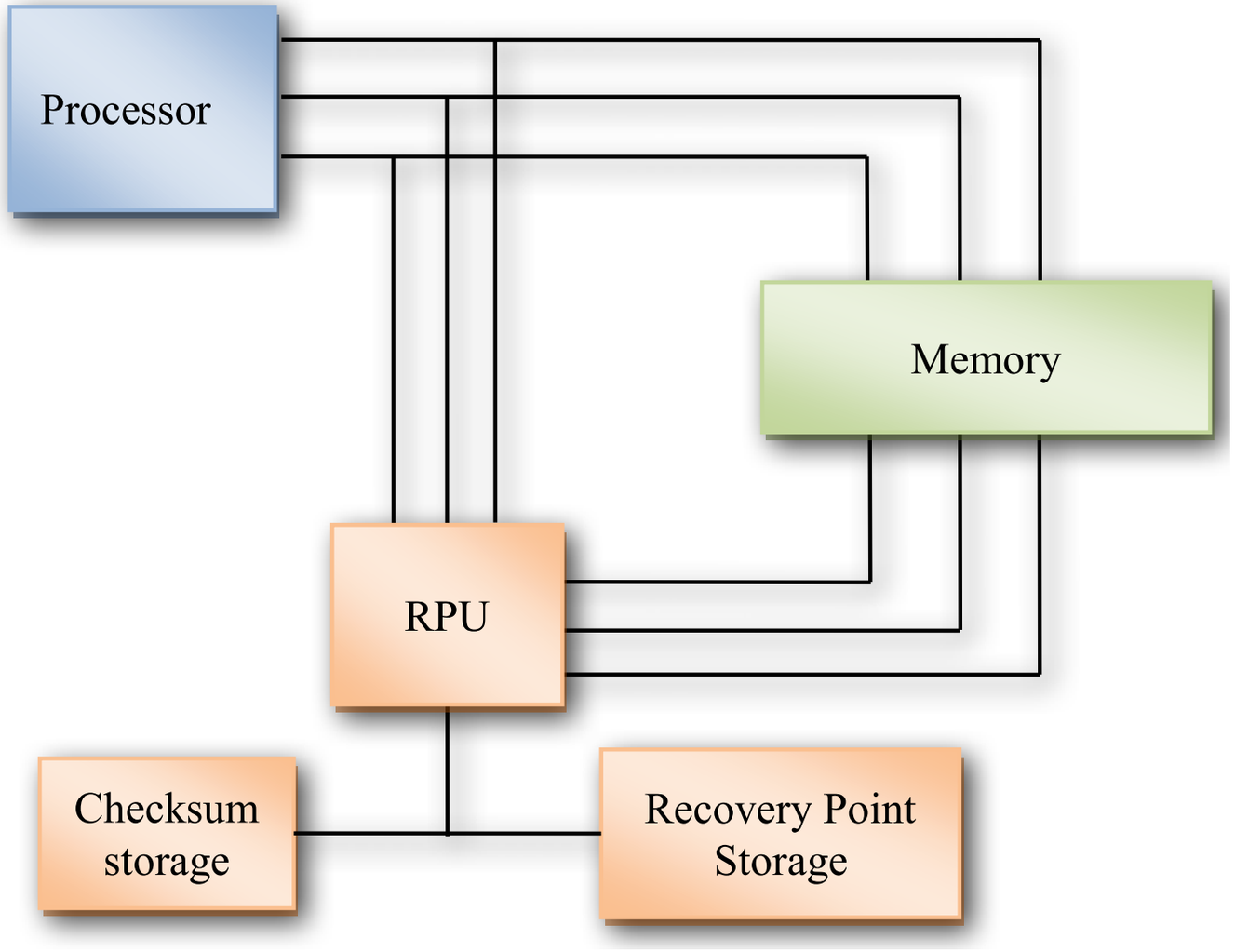
Hardware checking schemes must be involved in generation of a RP and CS for each program segment.

...The RP and associated CS should be generated concurrently with program execution.

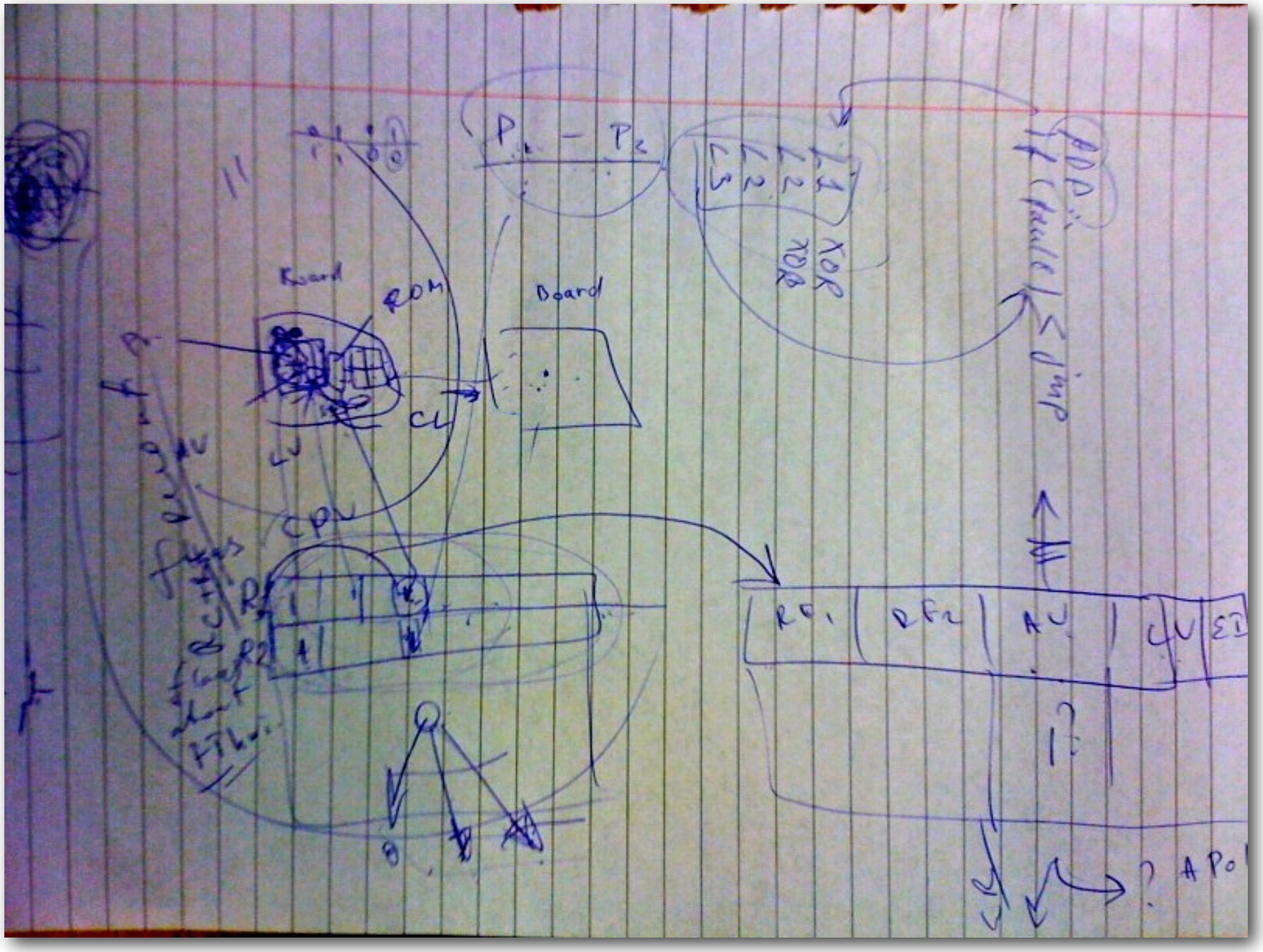
For performance gain the RP storage and checksum generator should be directly accessed by processor (to speed up RP generation and recovery).

When the error is detected, by direct intervention from Run-time system an access to RP sequence is enabled and searching of correct RP initiated.

...The checksum and recovery point device has two operating modes: the *standard mode* which creates RPs and CSs and *recovery mode* which generates only the checksums.
...The operating mode is configurable by the system software (Run-time system).



Reconfigurability: a syndrome for FT



File name: FT resolved, Sept 2010

First version of syndrome concept: witnessed by PhD students V Castano and A Petukhov.

Reconfigurability: a syndrome

Slightly better syndrome picture...

Power	Processor				Devices							Memory					
Power	CU	Registers	Arithmetic Unit	Logical Unit	Timer	Random Number Gen.	Interrupt Controller	Console	Stable Storage	UART1	UART2	UART3	ROM1	ROM2	RAM1	RAM2	RAM3	RAM4
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

From a system software point of view the syndrome is represented as a set of special hardware registers.

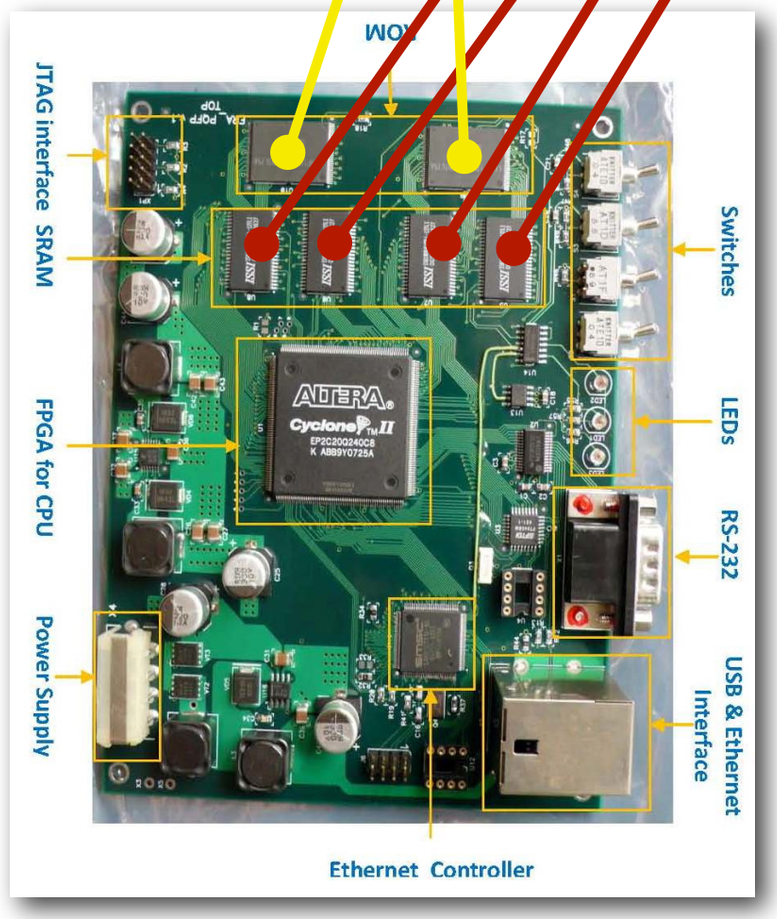
Syndrome Registers indicates the current hardware state (current configuration, detected faults, power)...

Fault detection schemes signal to syndrome causing hardware interrupts and initiation of GAFT by run-time system.

Run-time system, when necessary, executes reconfiguration of hardware.

Run-time system new functions of control are:

a) reconfiguration for reliability, performance or power-saving
b) control of graceful degradation



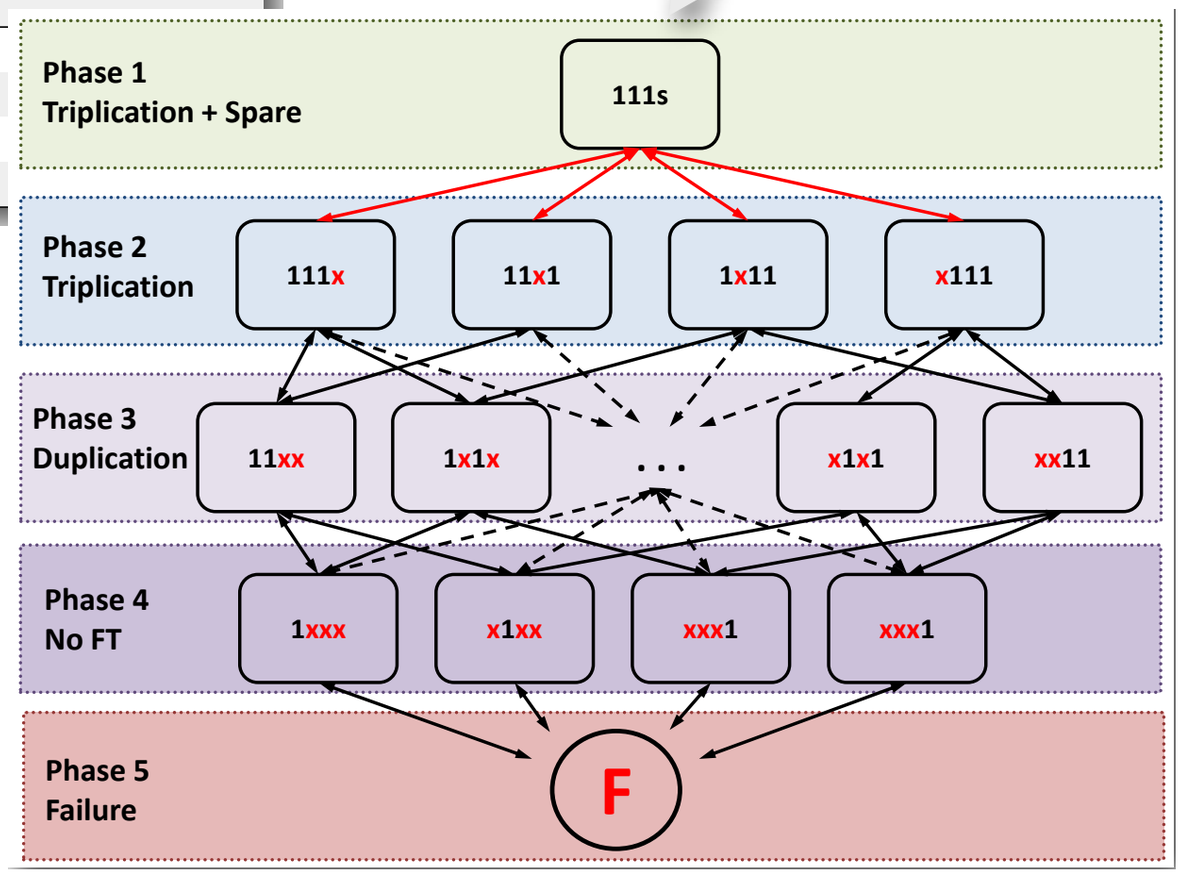
Reconfigurability: use of syndrome for memory

From defined by hardware design system configurations set memory configurations:

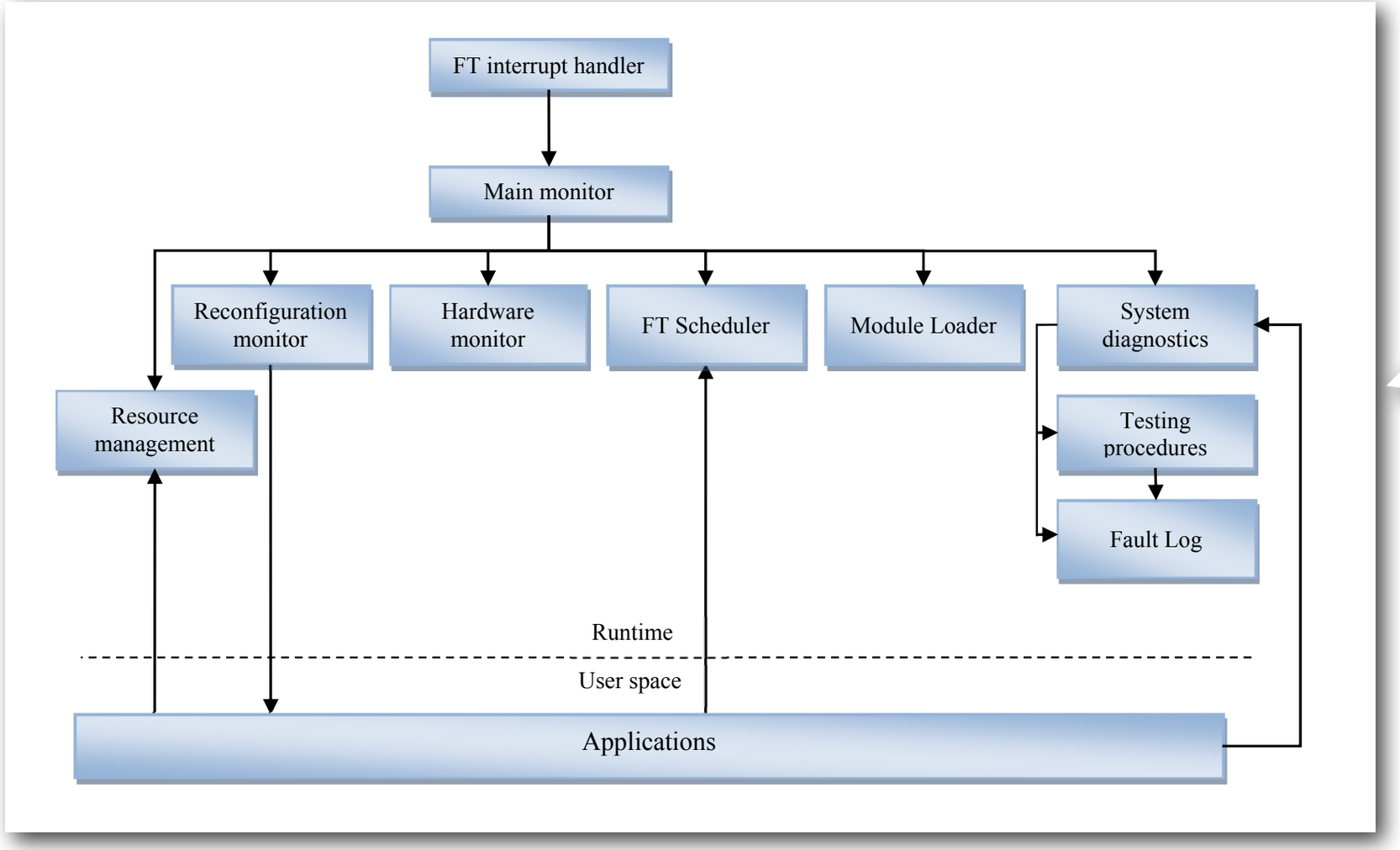
Mode Number	*Number of used banks*	*Redundancy Mode*	*Number of used memory modules*	*Usable Size in Mb*
1	1	Triplicated + 1 Spare	4	4
2	1	Triplicated	3	4
3	2	Triplicated + 1 Linear	4	8
4	1	Duplicated + 2 Spare	4	4
5	1	Duplicated + 1 Spare	3	4
6	2	Duplicated	4	8
7	3	Duplicated + 2 Linear	4	12
8	2	Duplicated + 1 Linear	3	8
9	4	Linear	4	16
10	3	Linear	3	12
11	2	Linear	2	8
12	1	Linear	1	4

Areas of processor, interfacing zone, passive zone in terms of configurations can be defined together with their degradation sequences. Configurations and their changes supported by run-time system, in principle, enabling sequential degradation “up to the last soldier”, when single element of each section left, but system will remains operable.

An example of system software control of memory degradation for triplicated memory

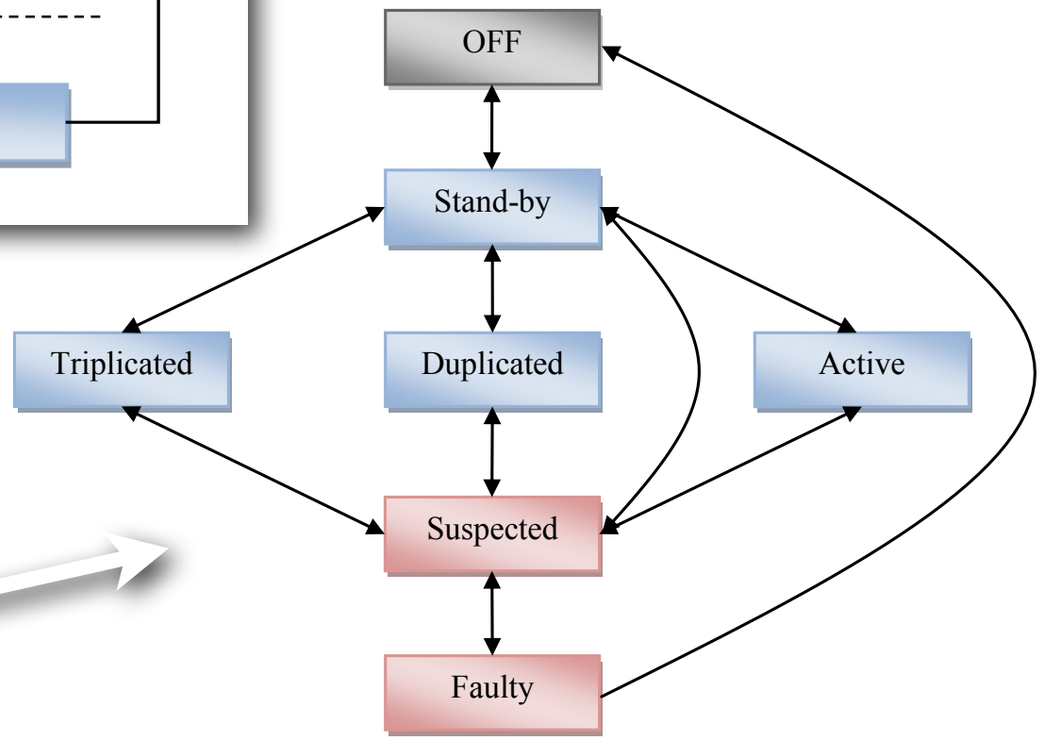


Runtime system for FT architecture

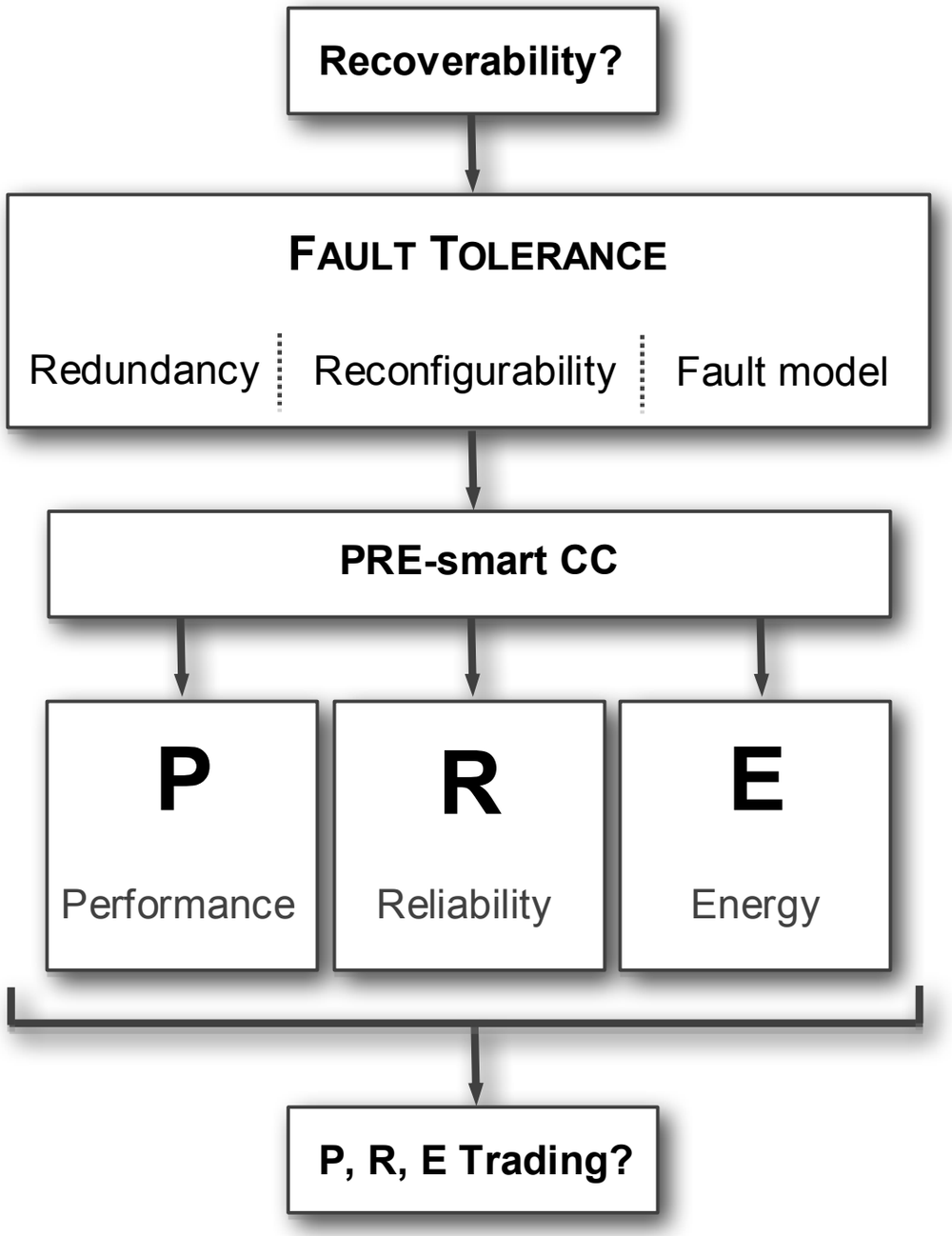


Runtime system architecture

Runtime system has to detect hardware fault, define fault type, handle hardware reconfiguration, control hardware degradation



Reconfigurability, FT, is it important? PRESSA



Redundancy and reconfigurability of a system can be exploited differently...

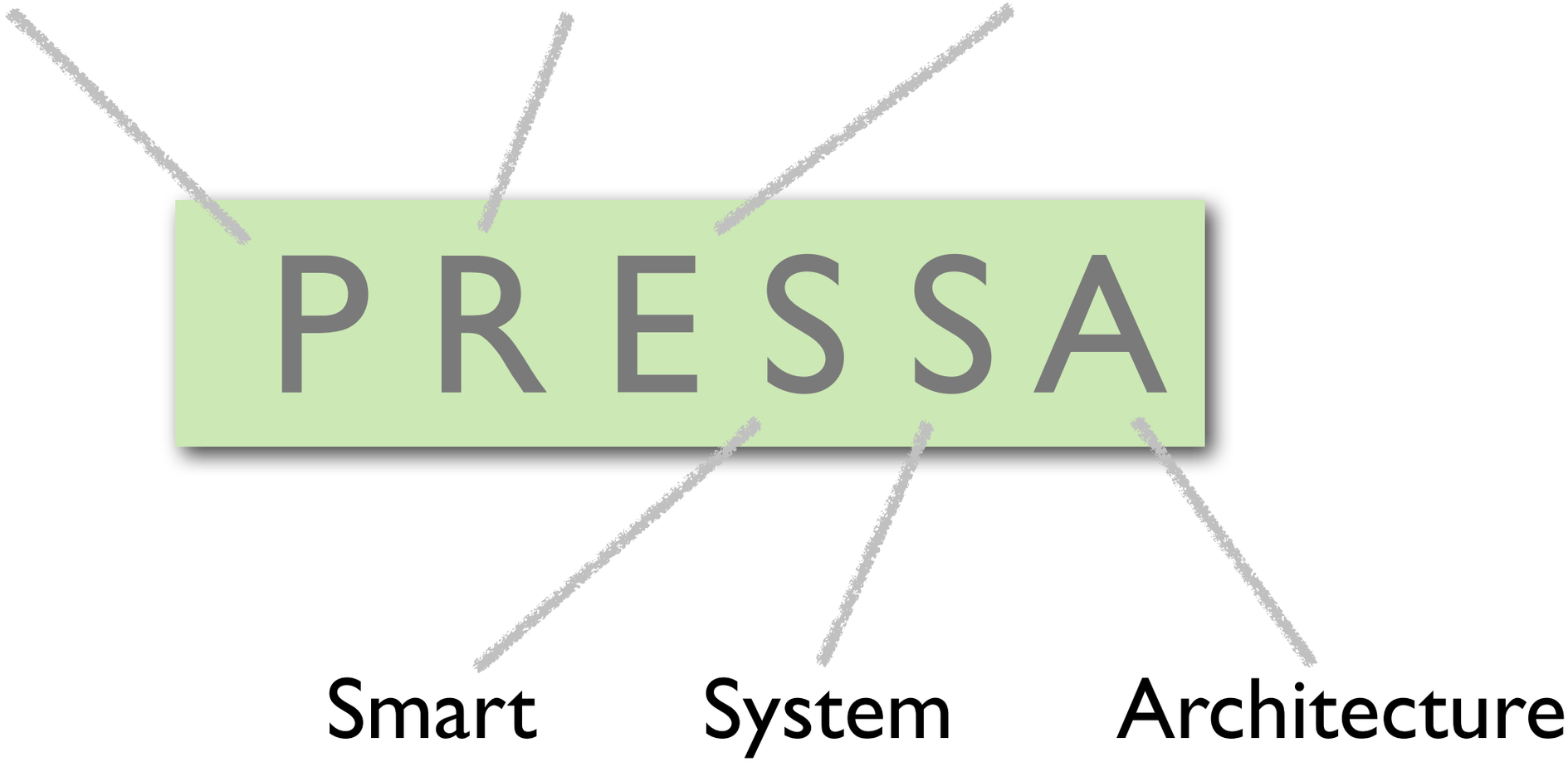
...gaining in:

Performance (P),
Reliability (R),
...or saving Energy (E).

Trading of P,R,E - in next generation of stand alone, connected and distributed systems is one of the biggest challenge...

Future: PRESSA concept

Performance Reliability Energy



More? → http://www.it-acs.co.uk/book.html

→ http://www.researchgate.net/profile/Igor_Schagaev/

Thanks for...

- Discussions, joint efforts: T Kaegi, S Monkman, B Kirk

- Discussions on redundancy: J C Laprie (late 80's)

- Discussions on reliability vs. FT: S Birolini (2005-10)

- Discussions on GLM: Felix Friedrich

- Pictures: S Monkman, V Castano

- Publication support: Chong Chen (Chinese version)

- Publication support: Yurzin Bogdanov (Russian version)

Materials used...

Book: T Kaegi, I Schagaev System Software Support of Hardware Efficiency, ITACS Ltd 2013, UK
http://www.it-acs.co.uk/book.html

Papers:

http://www.researchgate.net/publication/220908686_ERA_Evolving_Reconfigurable_Architecture
http://www.researchgate.net/publication/255701866_Greenwich_ERA_July_2010
http://www.researchgate.net/publication/253645564_Control_Operators_vs_Graph_Logic_Model
http://www.researchgate.net/publication/224370614_Reliability_of_malfunction_tolerance
http://www.researchgate.net/publication/252627186_Redundancy__Reconfigurability__Recoverability
http://www.researchgate.net/publication/244477145_Method_and_apparatus_for_active_system_safety
http://www.researchgate.net/publication/253643647_Redundancy_Classification_Principles_for_the_Design_of_Fault_Tolerant_Computers
http://www.researchgate.net/publication/252628898_Operating_system_for_fault_tolerant_SIMD
http://www.researchgate.net/publication/252629068_Using_Software_Recovery_For_Determine_The_Type_of_Hardware_Faults
http://www.researchgate.net/publication/253238741_Comparative_analysis_of_efficiency_of_algorithms_of_recovery_for_computing_process
http://www.researchgate.net/publication/252628869_Recovery_Points_And_Hardware_Reliability_Indices
http://www.researchgate.net/publication/252629150_Computing_process_restoration_algorithms
http://www.researchgate.net/publication/253584715_Using_Data_Redundancy_For_Program_Rollback
http://www.researchgate.net/publication/228728452_Hardware_Testing_on_the_Level_of_Tasks