Technische Universität Wien

# Predictable Timing on MPSoC
# A Time-Triggered View

Peter PUSCHNER

1st Workshop on Mapping Applications to MPSoCs Schloss Rheinfels, Germany June 2008

# Focus

goal: build safety-critical hard real-time systems

- that are distributed
- that have strict timing requirements
- adequate engineering process → simple concepts
  - support of construction
  - easy argumentation about properties (timing!)

# Hierarchical Design

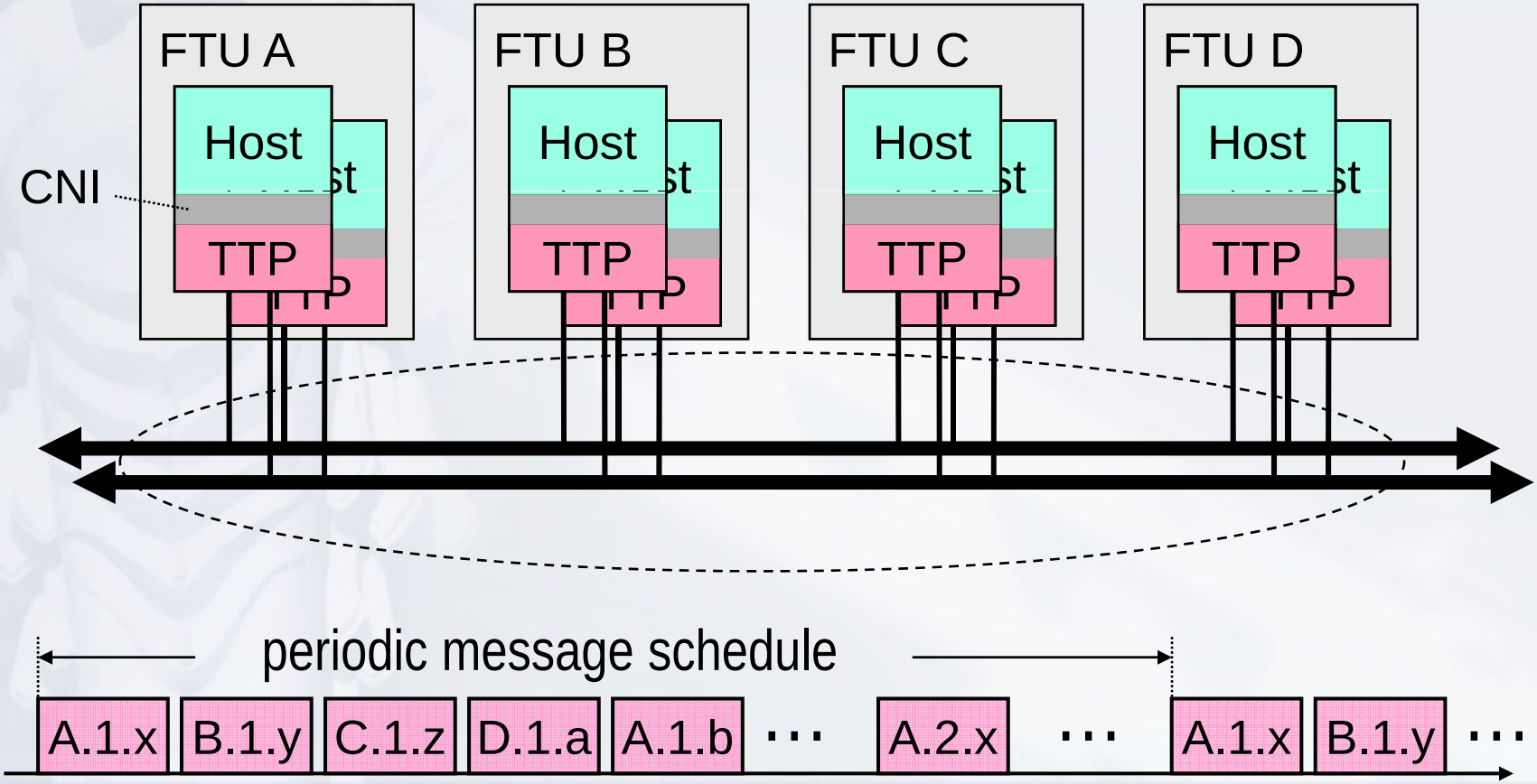
We need a hierarchical design to keep complexity manageable

- decomposition: spatial, temporal
- subsystems need to be (de)composable: weak interactions among subsystems

# What we need …

- simple, regular shape
  → dimensions are easy to assess, describe
- composability: it has the same dimensions under all circumstances (stand alone, when integrated, …)
- failures are easy to detect

# TTA System

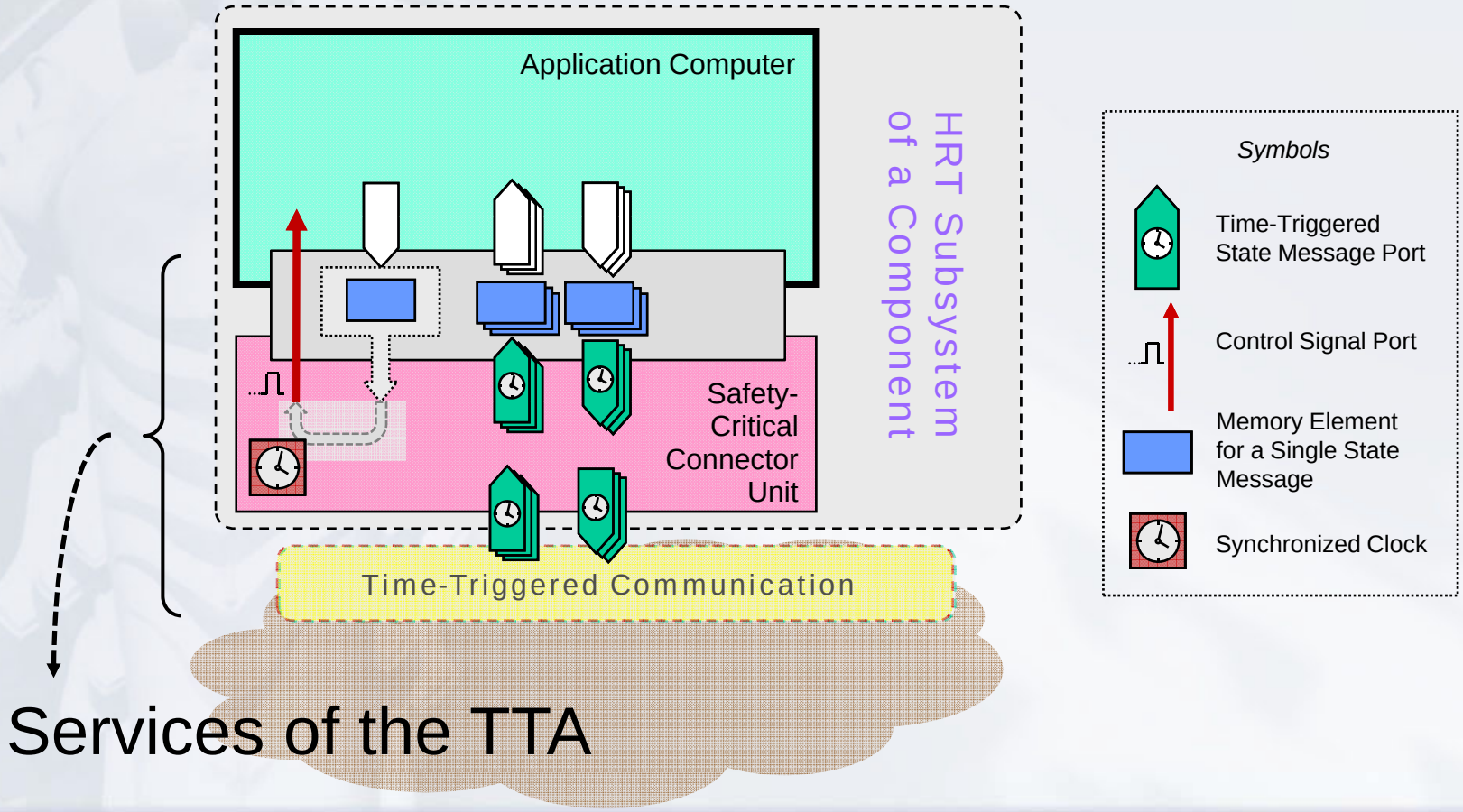

FTU A
FTU B
FTU C
FTU D
Host
TTP
CNI
periodic message schedule
A.1.x
B.1.y
C.1.z
D.1.a
A.1.b
A.2.x
A.1.x
B.1.y

# TTA Support for Composability

- Synchronized real-time reference clock
- Fixed bus schedule, planned before runtime
- Node interfaces are fixed before runtime
  → composability that facilitates a hierarchical design and evaluation

⇨ principle applicable to MPSoC:
offline planning of access to shared resources

# A Time-triggered HRT Subsystem



# Concerns at Component Level …

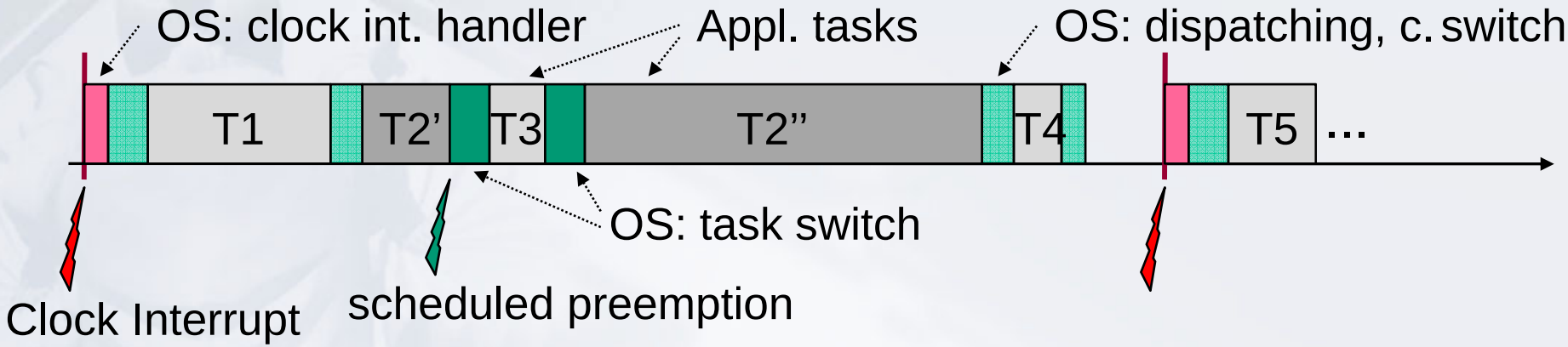
- Task timing
  - predictability
  - stability
  - composability – internal, external (no interference)
- Operating system
- Schedulability, scheduling

# OS Software and Scheduling

**Principle: take control decisions offline!!**

- Task model: simple tasks, single-path code
- Operating system structure & scheduling
  - Single-path code wherever possible
  - Static, table-driven scheduling:
    Offline decisions for I/O, comm., task switching and preemption
  - Use clock interrupt to synchronize with RT clock

# Static Table-Driven Schedule

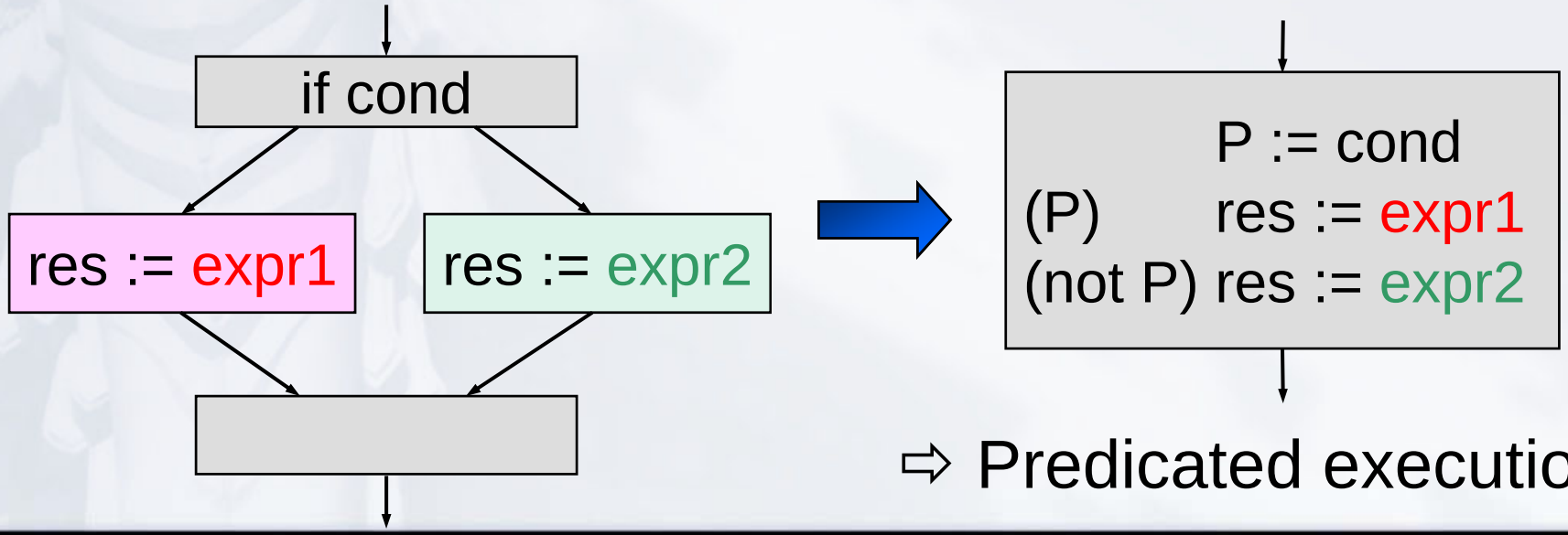



- Programmable clock interrupt
- Interrupt: start of defined task chains
- Statically scheduled preemptions
- Statically scheduled I/O and message access

# Single-Path Transformation

Transform input-data dependent branches into sequential predicated code, rest remains unchanged

⇨ Technique based on *if-conversion*




⇨ Predicated execution

# Single-Path Transformation Rules

Recursive transformation function based on syntax tree:

$$SP[\![ p ]\!]\sigma\delta$$

p … code construct to be transformed into single path

$\sigma$ … inherited precondition from previously transformed code constructs. The initial value of the inherited precondition is ‘T’ (logical true).

$\delta$ ... counter, used to generate variable names needed for the transformation. The initial value of $\delta$ is zero.

# Single-Path Transformation Rules (1)

simple statement: S

SP[[ S ]]σδ

➡

| | | |
|---|---|---|
| if σ = *T* : | S | // unconditional |
| if σ = *F* : | | // no action |
| otherwise: | (σ) S | // guarded |

# Single-Path Transformation Rules (2)

sequence: S = S1; S2

SP[[ S1; S2 ]]σδ

➡

*guard*$_\delta$ := σ;
SP[[ S1 ]]⟨guard$_\delta$⟩⟨δ+1⟩ ;
SP[[ S2 ]]⟨guard$_\delta$⟩⟨δ+1⟩

# Single-Path Transformation Rules (3)

alternative: S = if *cond* then S1 else S2 endif

SP[[ if *cond* then S1 else S2 endif ]]σδ

➡

if *ID*(cond):

$guard_\delta$ := *cond*;
SP[[ S1 ]]⟨σ ∧ $guard_\delta$⟩⟨δ+1⟩;
SP[[ S2 ]]⟨σ ∧ ¬$guard_\delta$⟩⟨δ+1⟩

otherwise:

if *cond* then SP[[ S1 ]]σδ
else SP[[ S2 ]]σδ
endif

# Single-Path Transformation Rules (4)

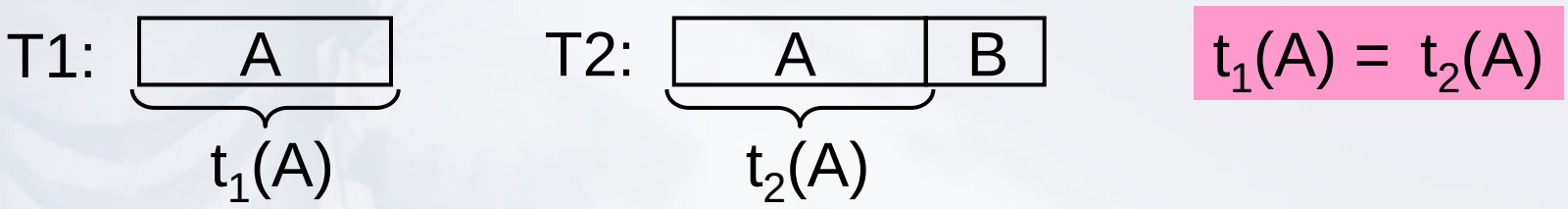
loop: S = while *cond* max *N* times do S1 endwhile

SP[[ while *cond* max *N* times do S1 endwhile ]]σδ

if *ID*(cond): $end_\delta$ := *F;* // loop-body-disable flag
for $count_\delta$ := 1 to *N* do // "hardwired loop"
SP[[ if ¬*cond* then $end_\delta$ := *T* endif ]]σ⟨δ+1⟩ ;
SP[[ if ¬$end_\delta$ then S1 endif ]]σ⟨δ+1⟩
endfor

# Composability of Timing

Composability (Sifakis): properties are conserved across integration




No guaranteed time composability in the presence of caches

Example: A, B in a loop; A fits into cache, but not AB; cache conflicts between instructions in A and B

# Composability of Timing (2)

Use prefetching (scratch pad memory) instead of caches to avoid side effects

Control flow of single-path code is determined at compile time

⇨ Exact pre-planning of prefetching (no speculation)

- ? Prefetch instructions in code
- ? Programmable prefetch controller

⇨ Composability + performance

# Conclusion

Mechanisms for stable, predictable MPSoC timing:

- Pre-scheduled access to shared resources (interconnects, shared memory)
- table-driven scheduling on cores
- use of single-path code
- explicitly controlled scratch-pad memories for speedup