Abstractions for indulgent distributed computing

R. Guerraoui
Distributed Programming Laboratory
Swiss Federal Institute of Technology in Lausanne (EPFL)

Summary

1. *There is no distributed computing middleware*
2. *It is challenging to devise one*
3. *Here is the story of our quest*

Roadmap of the talk

(1) Definitions

(2) Problem

(3) Proposition

Definition 1

A ***middleware*** is (1) a set of abstractions that implements a wide class of computing tasks and (2) a set of associated abstraction mechanisms

Examples of abstractions: *Set, list, record; semaphore, monitor*

Examples of abstraction mechanisms: *encapsulation, inheritance, interception*

Definition 2

A middleware for ***distributed*** computing is (1) a set of abstractions that solves a wide class of ***distributed*** computing tasks and (2) a set of associated abstracting mechanisms

Definition 3

A ***distributed computing task*** is one where ***several*** processes cooperate to achieve some ***common*** goal, despite the ***failure*** of a subset of the processes

Example

Distributed computing task **T**: processes exchange initial inputs to agree on one common output, despite crashes of some of the processes

Validity: the output is an input

Agreement: there is at most one output

Termination: there is at least one output

Claim

There is no middleware for distributed computing

Notes

Note 1. What has been called middleware so far is based on RPC-like ***centralized*** programming abstractions

Note 2. A lot of effort has been devoted to abstraction ***mechanisms*** but very little to the actual abstractions (Choices, Cactus, Garf and Bast, QuO,..)

Roadmap of the talk

(1) Definitions

(2) Problem

(3) Proposition

Problem

Devise a set ***X*** of abstractions for solving distributed computing tasks

Problem (cont'd)

X must be minimal and the abstractions be ***overhead-free*** and ***indulgent***

Overhead-freedom
1. Resilience

There should not be any solution to T using ***strictly weaker*** assumptions than those needed for X

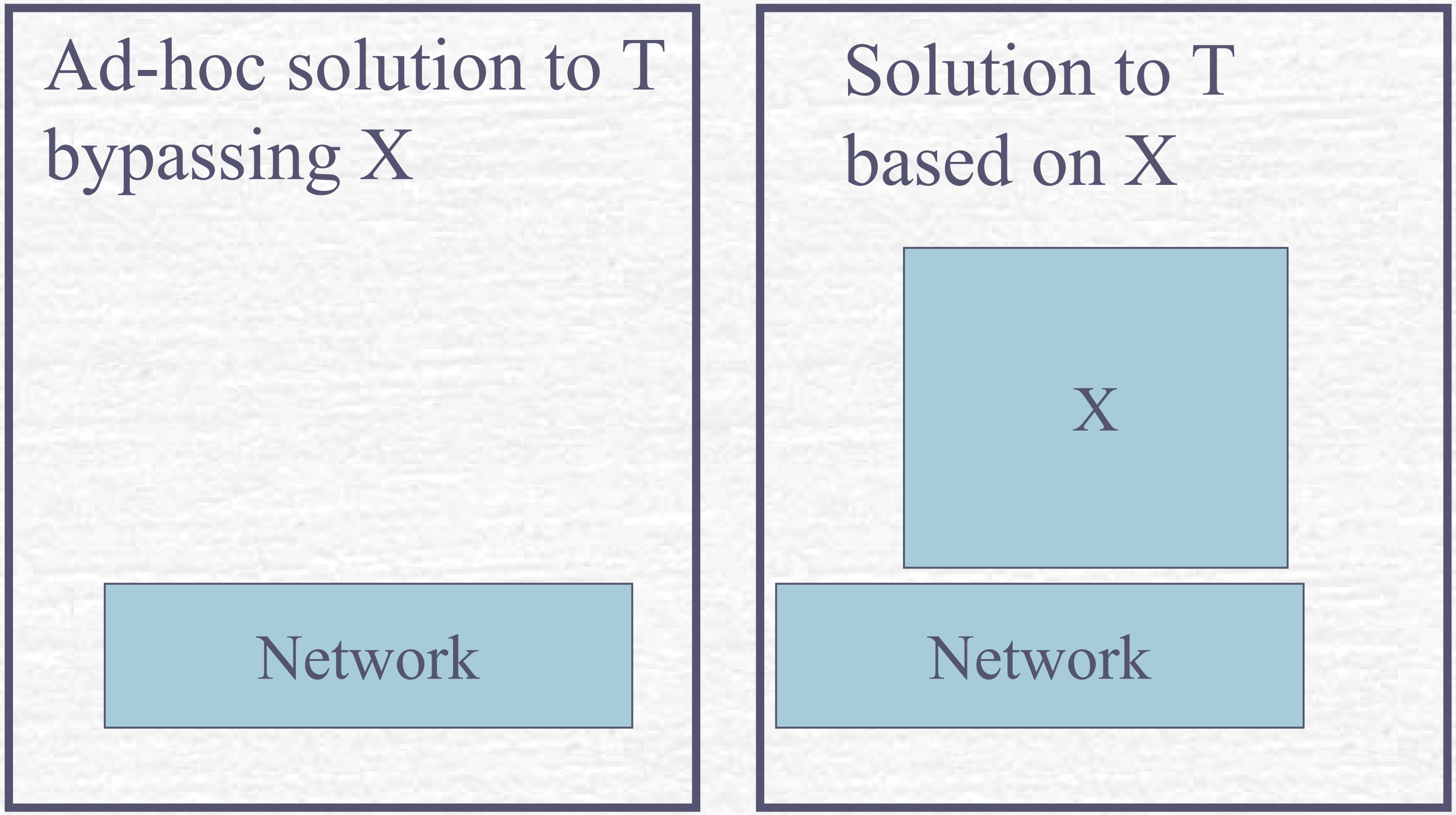
Example: X should not assume f+2 correct processes if some implementation of T assumes only f+1 correct processes

Overhead-freedom
2. Performance

No ad-hoc solution that bypasses X to solve T can be more ***performant*** than an X-based solution to T (with the same resilience)

Example: X should not inherently lead to solutions to T with 2n messages if some implementation of T needs only n messages

Overhead-freedom



Problem

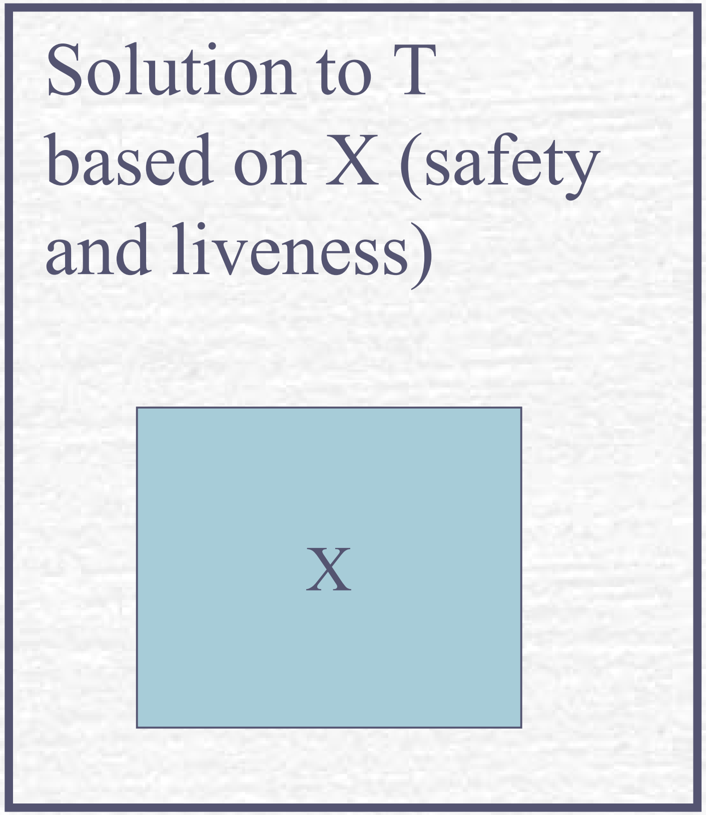
Devise a set ***X*** of abstractions for solving distributed computing tasks

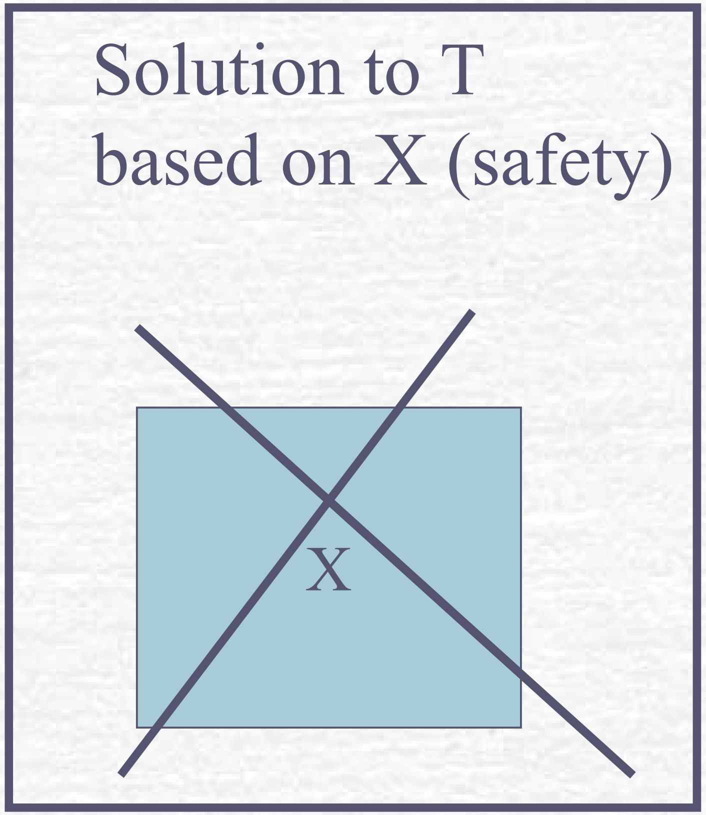
X must be minimal and the abstractions be ***overhead-free*** and ***indulgent***

Indulgence

Even if X does not comply with its specification, the ***safety*** of T is ensured

Indulgence


Solution to T
based on X (safety
and liveness)
X


Solution to T
based on X (safety)
X

Example (task T)

Processes exchange initial inputs to agree on one common output, despite crashes of some of the processes

Validity: the output is an input

Agreement: there is at most one ouput

Termination: there is at least one output

Why indulgence?

Because

« When they continued asking him, he lifted up himself, and said unto to them, ***He that is without sin among you, let him first cast a stone at her*** *» John 8:7(not Lennon)*

Why liveness?

Because

« While there is life there is hope » Cicero

Roadmap of the talk

(1) Definitions

(2) Problem

(3) Proposition

Proposition: X = {S,L}

S and L are the abstractions

S : A reliable form of storage

L : A reliable form of leader election

The S abstraction

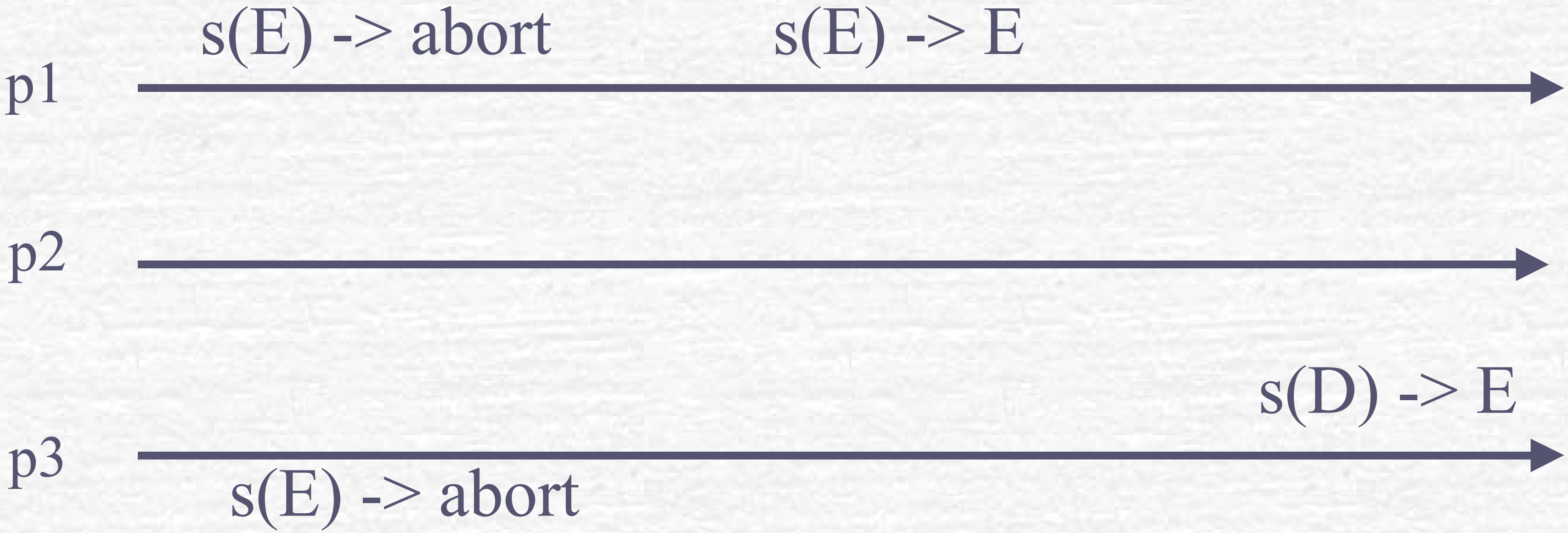
One operation ***s( )*** : s(value) -> {value',*abort*}

Two properties:

There can be at most one result ≠ *abort* and this must be an argument of s()

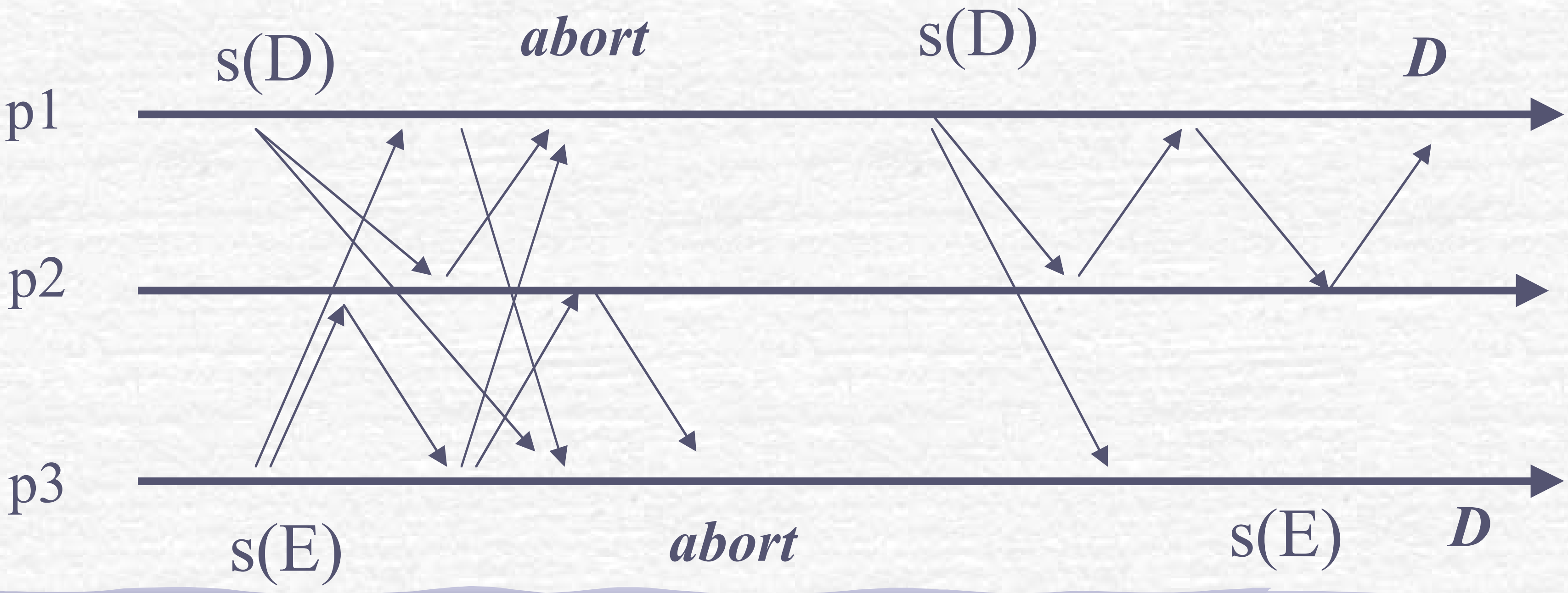
There must be a result and the result is *abort* only if two processes concurrently invoke s()

The S abstraction



Implementing S

S can be implemented in an asynchronous system with a majority of correct processes



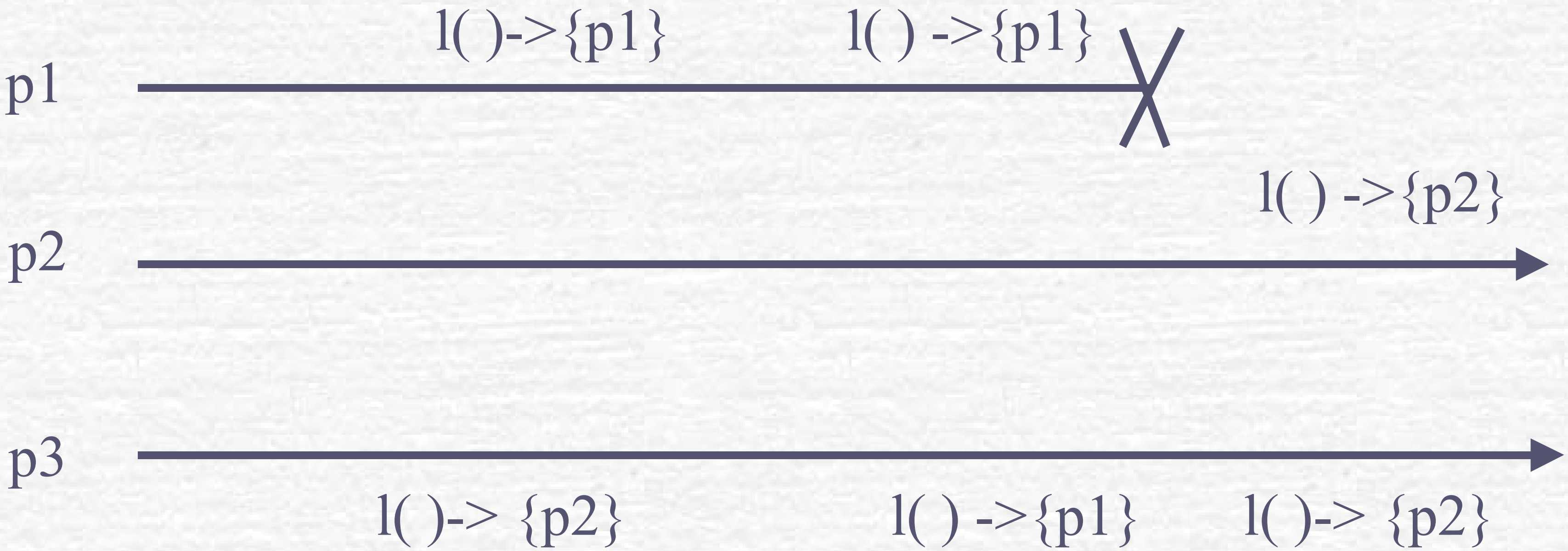
The L abstraction

One operation: ***l( )*** -> id

Property:

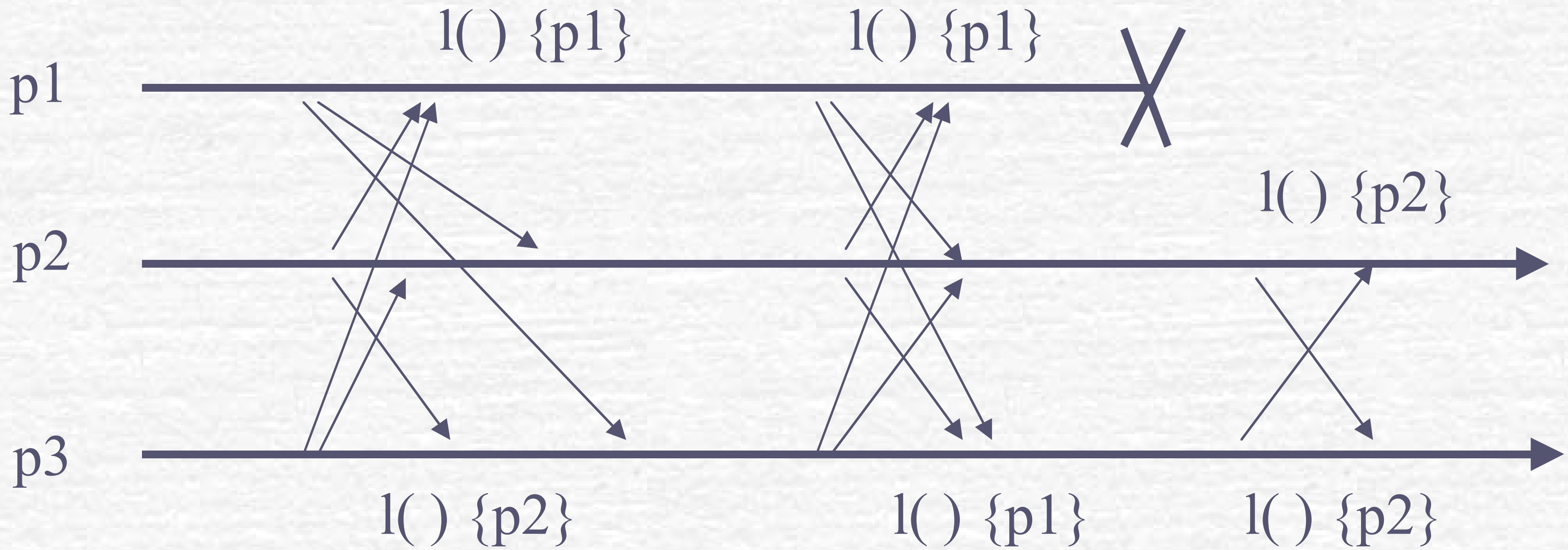
Eventually, the identity of one correct process is permanently returned

The L abstraction



Implementing L

L can be implemented in an eventually synchronous system



Remember our task T

Processes exchange initial inputs to agree on one common output, despite crashes of some of the processes

Validity: the output is an input

Agreement: there is at most one ouput

Termination: there is at least one output

Solution to T using X

Every process proposes an input and executes:

```
while true do
    if l() = self then
        if s(input) = v ≠ abort then
            return v
```

Indulgence

Any solution to T based on L is inherently indulgent (Gue:PODC00)

Overhead-freedom

Resilience

(1) L is minimal to implement T; (2) Using L, a correct majority (i.e., S) is needed to implement T (CT:PODC91; CHT:PODC92)

Performance

There is no indulgent solution to T that is more performant than the one using X (DFGP:DISC02; DG:PODC02)

Claims

S and L are convenient abstractions for distributed computing;

Not only for T and

Not only in a crash-stop model

(FG:DSN00;FG:PODC00;BDFG:DC03)

The fun is still ahead

What if we consider malicious processes?

What about timing issues?

What abstraction mechanisms?