

# Uncertainty and Predictability: can they be reconciled? *A Vision and a Concrete Model*

Paulo Veríssimo  
Univ. of Lisboa Faculty of Sciences  
Lisboa – Portugal

[pjv@di.fc.ul.pt](mailto:pjv@di.fc.ul.pt)

<http://www.navigators.di.fc.ul.pt>

---

# The Global Vision

# Problem Motivation

- Design and deployment of distributed applications is faced with the confluence of antagonistic aims:
  - ☞ between what is required by applications, and what is given by the supporting infrastructure/ environment
- Current and future large, massive-scale pervasive and/or ubiquitous computing systems will amplify this:
  - ☞ very high numbers of players, very large distances, geographical scope, topology and interconnections no longer a given, ill-defined COTS component properties
- Key lies with a changing notion of what should be the service guarantees:
  - ☞ about what have always been the fundamental issues, e.g., consistency, synchronism, reliability, availability, predictability, security, ...

# Grand challenges put by this scenario?

- Looks like a grand challenge would be withstanding uncertainty whilst achieving predictability
- **Uncertainty:**
  - ☞ is a common denominator of current systems
  - ☞ uncertain synchrony, fault model, and even topology
- **Predictability:**
  - ☞ systems are required to fulfil more and more demanding goals which imply predictability or determinism, e.g, timeliness, security
- **Reconciling them means:**
  - ☞ strong attributes can be secured in weak settings (e.g. on ordering, agreement, timely termination of algorithms), i.e. settings where usually very little is assumed and very little is expected from
  - ☞ current view has been to weaken attributes to the little one can expect from settings

# The Synchrony dimension

# Take the synchrony dimension

- Many services, beyond mere performance, have to secure **timeliness** properties, that is, they have to meet timing constraints (every x ms, within T, until T, etc.)
  - ☞ Dependability constraints: RT control applications; User-dictated QoS: Multimedia apps, synchronized groupware; Helper protocols: failure detection, clock synchronisation
- So we should use synchronous system models...
  - ☞ But unpredictably or unreliability of environment may make system fail
  - ☞ Dedicated infrastructures may be impractical or expensive
- Why not use asynchronous system models then?
  - ☞ They do not allow timeliness specifications

# The means and the goals...

- Goals - secure some safety and liveness properties
  - P1- any delivered message is delivered to all participants
  - P2- any message sent is delivered to at least one participant
- Means:
  - ☞ choice of model depends on efficiency and effectiveness of implementation, but above all on meeting the goals
- If we have Real-Time goals:
  - ☞ some safety properties are *timeliness* properties (even if one!)
    - P3- any delivered message is delivered within  $T_{Dmax}$  from the time of send request
  - ☞ model must represent time in some form (sync, timed part-sync, hybrid)

		<b>MEANS</b>	
		t	<del>t</del>
<b>GOALS</b>	T	✓ (SomeSync)	---
	<del>T</del>	✓ (SomeSync)	✓ (ASync)

T - timing variables in properties (timeliness goals, ex. delay bound)

t - timing variables of the support environment (timed means, ex. timeouts, clocks)

# "All systems are synchronous"

- synchrony appears where you least expect it, even in models having nothing to do with RT:
- **Asynchronous with Failure Detectors (Chandra/Toueg):**
  - ☞ a perfect crash FD runs in a synchronous environment
- **Timed Asynchronous Model (Cristian/Fetzer):**
  - ☞ real-time clocks are synchronous devices
  - ☞ a hardware watchdog is a synchronous device

T - timing variables in properties (timeliness goals, ex. delay bound)

TA - timing variables (often hidden) in auxiliary properties (timeliness reqs., ex. WDog or pCFD)

t - timing variables of the environment (timed means, ex. tout, clock, tx delay)

		MEANS	
		t	<del>t</del>
GOALS	T	✓ (SomeSync)	---
	TA	✓ (SomeSync)	---
	<del>T</del>	✓ (SomeSync)	✓ (Async)

# "All systems are synchronous"

- synchrony appears where you least expect it, even in models having nothing to do with RT:
- **Asynchronous with Failure Detectors** (Chandra/Toueg):
  - ☞ a perfect crash FD runs in a synchronous environment
- **Timed Asynchronous Model** (Cristian/Fetzer):
  - ☞ real-time clocks are synchronous devices
  - ☞ a hardware watchdog is a synchronous device

is there a unifying model?

		MEANS		
		T	t	
GOALS	T	✓	✓	---
	T <sub>A</sub>	✓	✓	---
	T <sub>✓</sub>	✓	✓	✓ (Async)

T - timing variables in properties (timeliness goals, ex. delay bound)

T<sub>A</sub> - timing variables (often hidden) in auxiliary properties (timeliness reqs., ex. WDog or pCFD)

t - timing variables of the environment (timed means, ex. tout, clock, tx delay)

# Why a new model

- Models of intermediate synchrony have been around:
  - ☞ Asynchronous with Failure Detectors (Chandra/Toueg)
  - ☞ Timed Asynchronous Model (Cristian/Fetzer)
  - ☞ Partially synchronous (Dolev, Dwork)
  - ☞ Immersion (Herman/LeLann)
  - ☞ Quasi-Synchronous Model (Verissimo/Almeida)
- None of the partial synchrony models is generic enough
  - ☞ each model treats synchrony asymmetries in its own way, relying on the evolution of synchrony with time, or with space, or both
  - ☞ none can handle the whole synchrony spectrum
  - ☞ they tie application styles & semantics in one way or another

## Why a new model

- The models of intermediate synchrony solved the problem only partially 😊

A common feature we observed in these works: synchronism (or asynchronism) are not homogeneous properties of systems--- they vary with time and with space, i.e. the part of the system being considered

- *The question is: how to control this process to our benefit?*

# The conceptual “SYNC” toolbox

- Assume that we try to find the simplest and at the same time most useful synchrony tools to build systems of any “Sync/ParSync/Async” flavour
- We propose to start with:
  - ☞ TE – timely execution (simple or generic)
  - ☞ DM – duration measurement (local or distributed)
  - ☞ CFD – crash failure detection (local or distributed)
  - ☞ TFD – timing failure detection (local or distributed)

# Alternative models

- Observations:
- Weakest of known timed system is TA, but no pFD possible on strict TA
- Hypotheses:
- Consider **TA+SYNC**(*simpleTE*, *localDM*), simplest set of services
- Consider applications where slow nodes do *harakiri*
- Thesis:
  - ☞ Tout => Crash Failure (CF)
  - ☞ Tout requires localDM
  - ☞ *enforce*(Crash) requires simpleTE
  - ☞ this simpleTE can be implemented with a watchdog (WD)
  - ☞ pCFD is achieved, through the transformation  $Tout \Rightarrow enforce(Crash)$ , and some algorithms to ensure that any P crashes before being suspected [Fetzer01]

# Alternative models

- Observations:
- For applications that survive timing faults in one process  $T_{out} \Rightarrow$  Crash Failure is not sufficient
- WD is a mere 'click': when fail-safety requires orderly shutdown routines this is not sufficient
- A fully synchronous subsystem is required to implement a pCFD [CT96]
- Interesting applications for unrestrained failure modes require a perfect crash failure detector [DelporteFauconnierGuerraoui02]
- Hypotheses:
- Consider **Async+SYNC**(*genericTE*, *distrDM*)
- Consider all interesting reliable non-timed applications
- Thesis:
  - ☞ with *distrDM* and *genericTE* we can build *distr pCFD* inside the SYNC box
  - ☞ Now we have **Async+SYNC**(*pCFD*), that is, ChandraToueg systems with pCFD
  - ☞ Of course, we can also have **TA+SYNC**(*genericTE*, *distrDM*, *pCFD*)

# Alternative models

- Observations:
  - To build timely applications, even soft, **TA** even with a strong **SYNC** box, is not sufficient, since TA itself is not timely
- Hypotheses:
  - Consider **RT+SYNC**(*genericTE*, *distrDM*, *pCFD*)
  - Consider all interesting reliable timely applications
- Thesis:
  - ☞ with **RT+SYNC**(*genericTE*, *distrDM*, *pCFD*) we can build timely appl's,
  - ☞ but we can not build reliable appl's if there are timing failures

# Alternative models

- Observations:
  - If we have timing failure detection we can build all interesting reliable timely applications [VerissimoCasimiro99]
  - If the SYNC box concentrates all crucial time services, then the (rest of the) system can have any synchrony, without that implying a change in model
- Hypotheses:
  - Consider **AnySyn+SYNC**(*genericTE*, *distrDM*, *pCFD*, *pTFD*)
  - Consider all interesting reliable applications, timely or not even timed
- Thesis:
  - ☞ with **SYNC**(*genericTE*, *distrDM*, *pCFD*, *pTFD*) we can detect timing failures and recover from them
  - ☞ we can build any applic. from time-free to RT with this SYNC box, depending on the properties of the rest of the system
  - ☞ more, we can build and make coexist applications and protocols with several flavours of timeliness

# 'SYNC' box as a generic framework

- **SYNC**(*TE, DM, CFD, TFD*) is the “toolbox” we should aim for:
  - ☞ **TE** – timely execution (simple or generic)
  - ☞ **DM** – duration measurement (local or distributed)
  - ☞ **CFD** – crash failure detection (local or distributed)
  - ☞ **TFD** – timing failure detection (local or distributed)
- Given the concept, we can instantiate as simple SYNC-box based systems as need be:
  - ☞ **TA+SYNC**(*simpTE (WD), locDM*)
  - ☞ **Async+SYNC**(*genTE, distrDM, pCFD*)
  - ☞ **TA+SYNC**(*genTE, distrDM, pCFD*)
  - ☞ **RT+SYNC**(*genTE, distrDM, pCFD*)
  - ☞ **AnySyn+SYNC**(*genTE, distrDM, pCFD, pTFD*)

- If we had such a configurable box, we could design systems and algorithms for several flavours of synchrony, using the same underlying paradigms. For example, see figure:

- ☞ SYNC toolbox (strong green) is the only sync part in the whole system, it secures whatever  $T_A$  auxiliware hard sync properties required, by supplying the respective 't' parameters in a trustworthy way
- ☞ Systems macroscopically asynchronous (orange) can in fact be designed in a time-free way (not-t), because they do not have timeliness requirements (not-T). The SYNC configuration should aim at supplying just the needed auxiliware  $T_A$  to guarantee a correct design (ex. implement WD or pCFD)
- ☞ When the design requires timeliness, which may take increasingly strong forms, i.e. of partial synchrony or equivalently real-time (from orange to green), the T properties must be secured by a combination of the fundamental and reliable SYNC functions (strong green) and the sheer environment timeliness (several shades of green, depending on how hard RT), together they contribute to the required 't' parameters

		MEANS	
		t	<del>t</del>
GOALS	T	(SomeSync)	---
	$T_A$	(Sync)	---
	<del>T</del>	---	(Async)

# Are we done?

# How to build a system with T (timeliness) properties?

- Observations:
- The relevant issues are: achieving 't' (low-level) parameters; making them available; obtain  $T_p$  (high-level) timeliness properties
- Hypotheses:
- Consider layer with 't' parameters, and service/protocol 'P' which uses 't' directly, deriving property-level  $T_p$  [Cristian $\Delta$ t85, KopetzTTP93]
- Thesis:
  - ☞ With a synchronous layer offering the 't' parameters, we build synchronous protocols that rely on 't', offering services with timeliness properties  $T_p$

# How to build a system with T (timeliness) properties?

- Observations:
- If 't' is violated, P fails, sometimes not only timeliness properties Tp, but also safety properties
  - ☞ (ex. order inversion caused by late messages in total order  $\Delta t$  prots; message corruption caused by clock skew exceeded in TDMA TT prots)
- Hypotheses:
- Consider environment with 't' parameters, and service/protocol 'P' whose Tp properties are implicitly indexed to 't'
- Thesis:
  - ☞ P may be time-free by construction, and there is a time complexity equation indexed to 't'
  - ☞ Each time P is *immersed* in a target environment, actual values are derived for Tp, depending on 't' [HermantLeLann, Delta4Amp]

# Achieving Synchronism by immersion

- «How to achieve synchronism with a timer-driven (“async”) protocol, given that classical approaches to synchronous protocols are clock-driven (“sync”)?:
  - ☞ achieve and determine upper bounds on frame delivery delays by the abstract network, in the presence of load and faults (T<sub>td</sub>);
  - ☞ impose a performance specification on the NAC hardware/software (CPU, kernel, etc.) in order that processing times of the protocol actions are bounded, and known for the worst case traffic pattern specified;
  - ☞ structure the protocol in phases, so that an execution predictably has a bounded number of phases; clearly delimit phases, in what concerns error detection/recovery (omission and timing), and permanent failure detection (exceeding assumed bounds);
  - ☞ structure each phase as a series of timed-out transmissions-with-response, so that it can be decomposed in time, in a sequence of frame deliveries and protocol actions as specified above, having thus with a known duration bound.» [\[VerissimoSRDS90\]](#)
- **This very early attempt was used in the Delta4 Amp: delivery time would be bounded to a known value, despite the acked-based, non clock-driven structure of the protocol**

# Achieving Synchronism by immersion

- «How to achieve synchronism with a timer-driven (“async”) protocol, given that classical approaches to synchronous protocols are clock-driven (“sync”)?:
  - ☞ achieve and determine upper bounds on frame delivery delays by the abstract network, in the presence of load and faults (T<sub>td</sub>);
  - ☞ impose a performance specification on the AC hardware/software (CPU, kernel, etc.) in order that processing times of the protocol are bounded and known for the worst case traffic pattern specified;
  - ☞ structure the protocol in phases, so that an execution predictably has a bounded number of **TIME-FREE** permit phases, in what concerns error detection/recovery (omission and timing), and permit **PROTOCOL STRUCTURE** (time-limited bounds);
  - ☞ structure each phase as a series of timed-out transmissions with response, so that it can be decomposed in time, in a sequence of frame deliveries and protocol actions as specified above, having thus a **IMMERSION** bound.»

- This very early attempt comes from the Delta4 AMP: delivery time would be bounded to a known value, despite the acked-based, non clock-driven structure of the protocol [VerissimoSRDS90]

# Pitfalls of immersion

- Observations:
- When 't' varies, Tp also varies
- Protocols which have no strict timeliness properties will move faster or slower, depending on 't'; they will always be safe **if they are time-elastic** [VerissimoCasimiro99]
- Protocols with timeliness properties Tp (real-time) indexed by immersion to a given magnitude of 't', will give timing failures when 't' increases, **violating timeliness, and maybe safety by contamination** [VerissimoCasimiro99]
- Protocols with safety properties that depend on the environment synchronism ('t' being constant) **will fail violating safety if 't' is exceeded** [VerissimoCasimiro99]
- Hypotheses:
- Consider environment with 't' parameters, and service/protocol 'P' whose Tp properties are implicitly indexed to 't'
- Thesis:
  - ☞ When 't' increases, Tp is violated, giving a timing failure. If it is not detected and processed, contamination may arise and lead to safety properties failure
  - ☞ If the protocol is a pCFD, it needs a sync system. If the bounds are violated, the CFD fails

# Architecture and coverage

- Immersion is a good concept, but of limited utility
- For example, in the past discussion, there were two crucial protocols whose safety requires a fully sync environment ('t' constant or bounded): pCFD and pTFD
- So architecting these protocols just using immersion on the same synchronous environment used to support (provide 't' normal (payload) protocols, leads to a coverage problem:
  - ☞ Suppose that the whole system is complex: the synchronous layer becomes complex, since it has the same footprint, and coverage comes down
  - ☞ If 't' parameters increase, whilst time-elastic applications may adapt ok, certainly the protocols above (auxiliware) will fail
  - ☞ So even if we do not have applications with rigid timeliness properties (giving timing failures), the immersion system is limited
  - ☞ **How do we enforce 't' for complex systems?** ☹

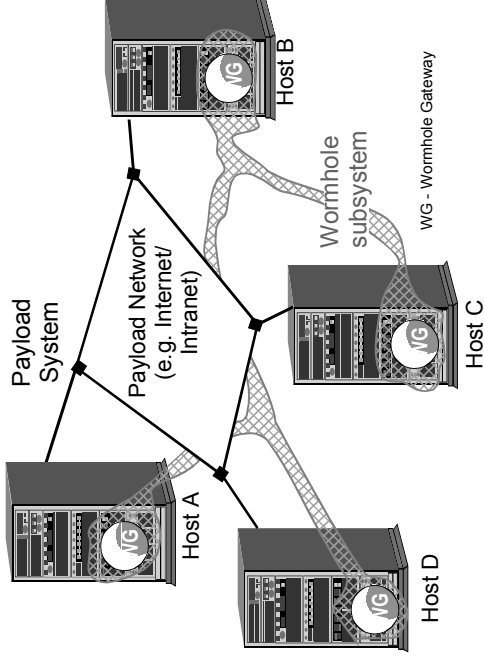
# Architecture and coverage

- Suppose the system is hybrid with regard to faults and synchrony
  - ☞ **Payload**: any synchrony, indulgent w.r.t. ‘tp’ of the environment
  - ☞ **Control**: fully asynchronous, strict w.r.t. ‘tc’ of the environment
  - ☞ It is built through **architectural hybridization**:
    - the part of the environment supplying ‘tc’ is specially built: we do get ‘tc’!
    - the rest (payload) is normal stuff: ‘tp’ is not so assured, or not at all
- Consider building a **SYNC** toolbox inside the **control** part
  - ☞ Implement the SYNC functions in control part, in this case, pCFD, pTFD
- Immerse payload algorithms (e.g. time-free) into the ‘tp’ environment.
- ‘tp’ hypotheses may fail, but do not affect the FDs
- Besides, ‘tp’ hypotheses failure can be detected by control (ex. pTFD) with high assurance (given by ‘tc’), allowing to support other than time-elastic protocols

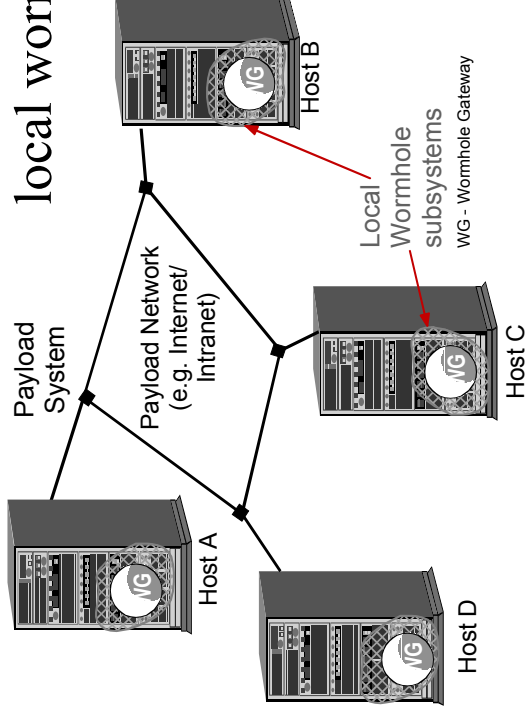
# Wormholes

- New design philosophy for distributed systems:
- constructs with privileged properties which endow systems with the **capability** of **evading the uncertainty** of the environment (“taking a shortcut”) for certain crucial steps of their operation, in order to achieve the required “hard properties” (predictability)

distrib. wormholes

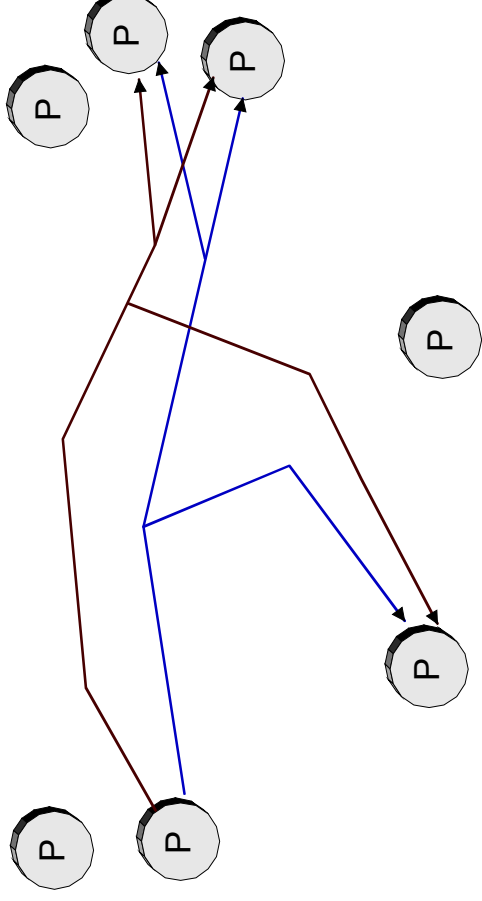


local wormholes



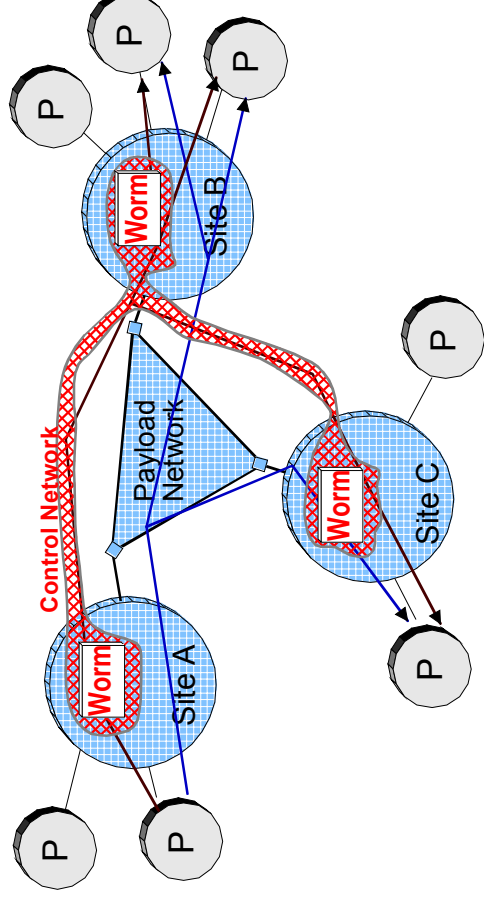
# Wormholes seen from inside

- We have played recently with two types of wormhole subsystems, to prove the concept:
  - ☞ Timely Computing Base for timeliness
  - ☞ Trusted Timely Computing Base for timeliness and security



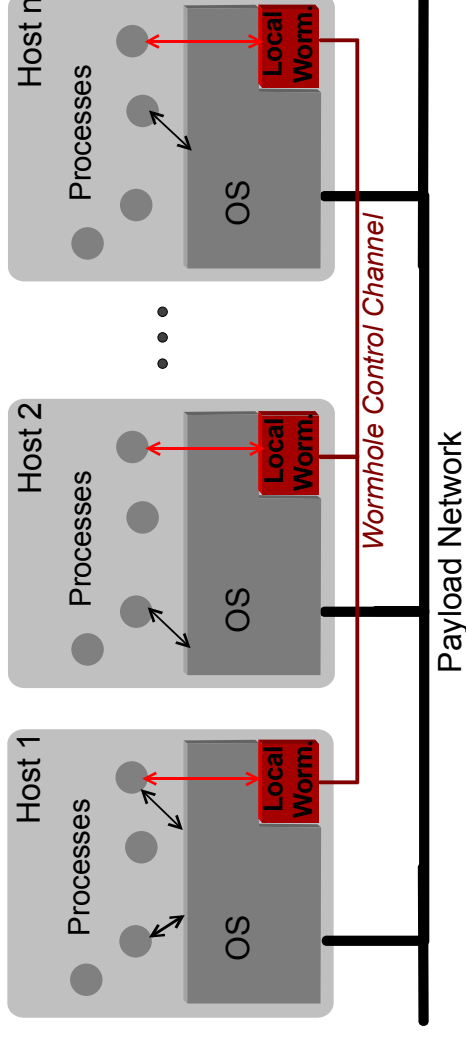
# Wormholes seen from inside

- We have played recently with two types of wormhole subsystems, to prove the concept:
  - ☞ Timely Computing Base for timeliness
  - ☞ Trusted Timely Computing Base for timeliness and security

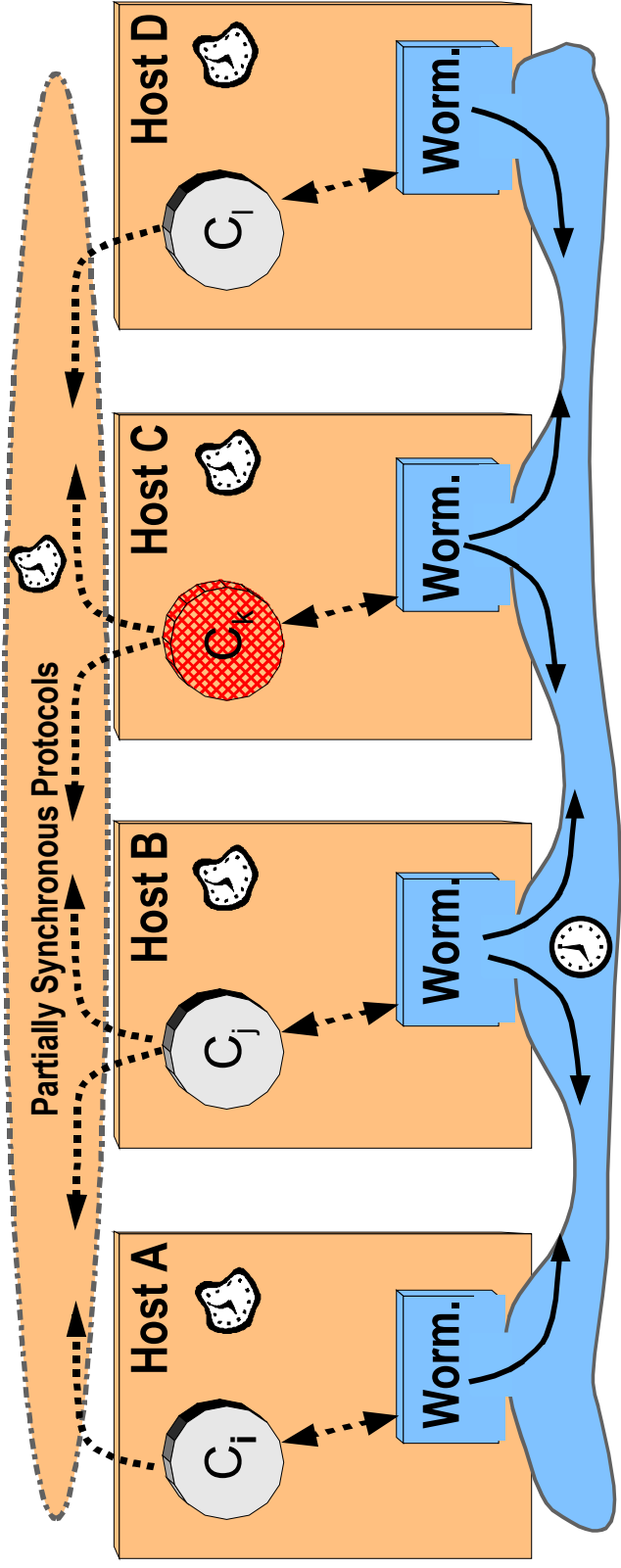


# Wormholes seen from inside

- We have played recently with two types of wormhole subsystems, to prove the concept:
  - ☞ Timely Computing Base for timeliness
  - ☞ Trusted Timely Computing Base for timeliness and security

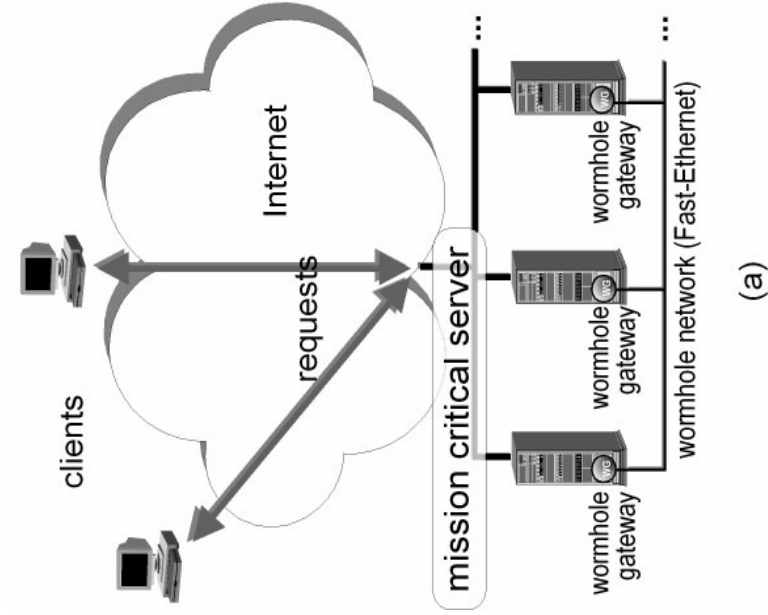


# Strategy for timeliness awareness and/or assurance

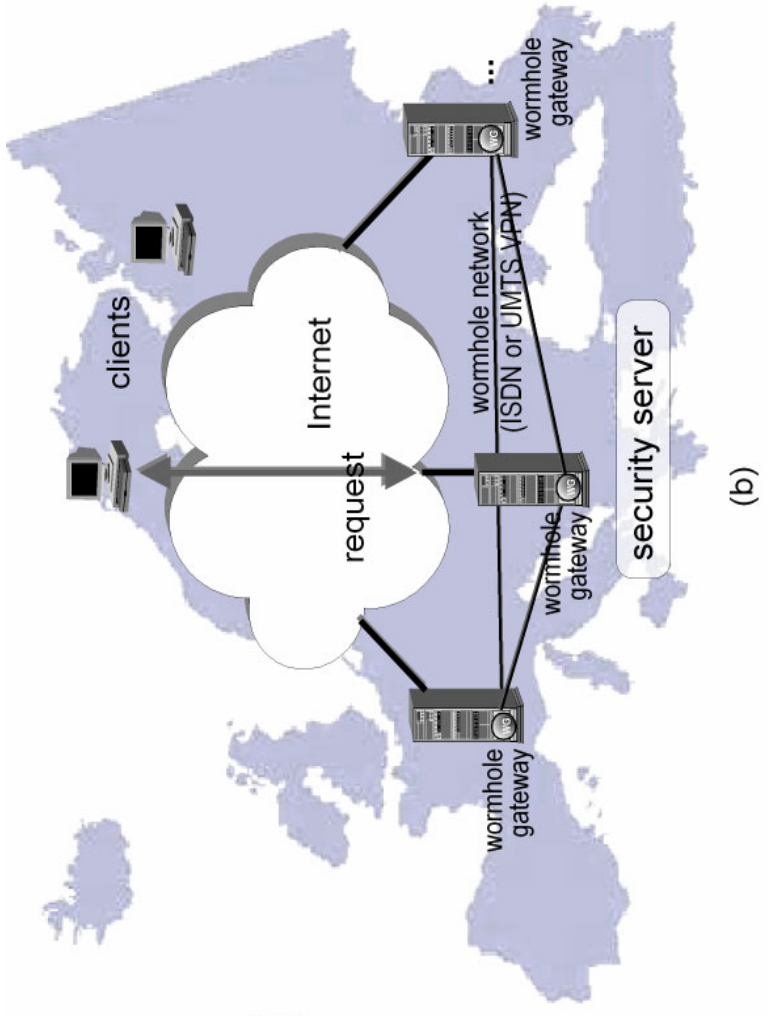


- - Fully synchronous, timely
- - Partially synchronous, potentially untimely

# Example of deployment of systems with wormholes

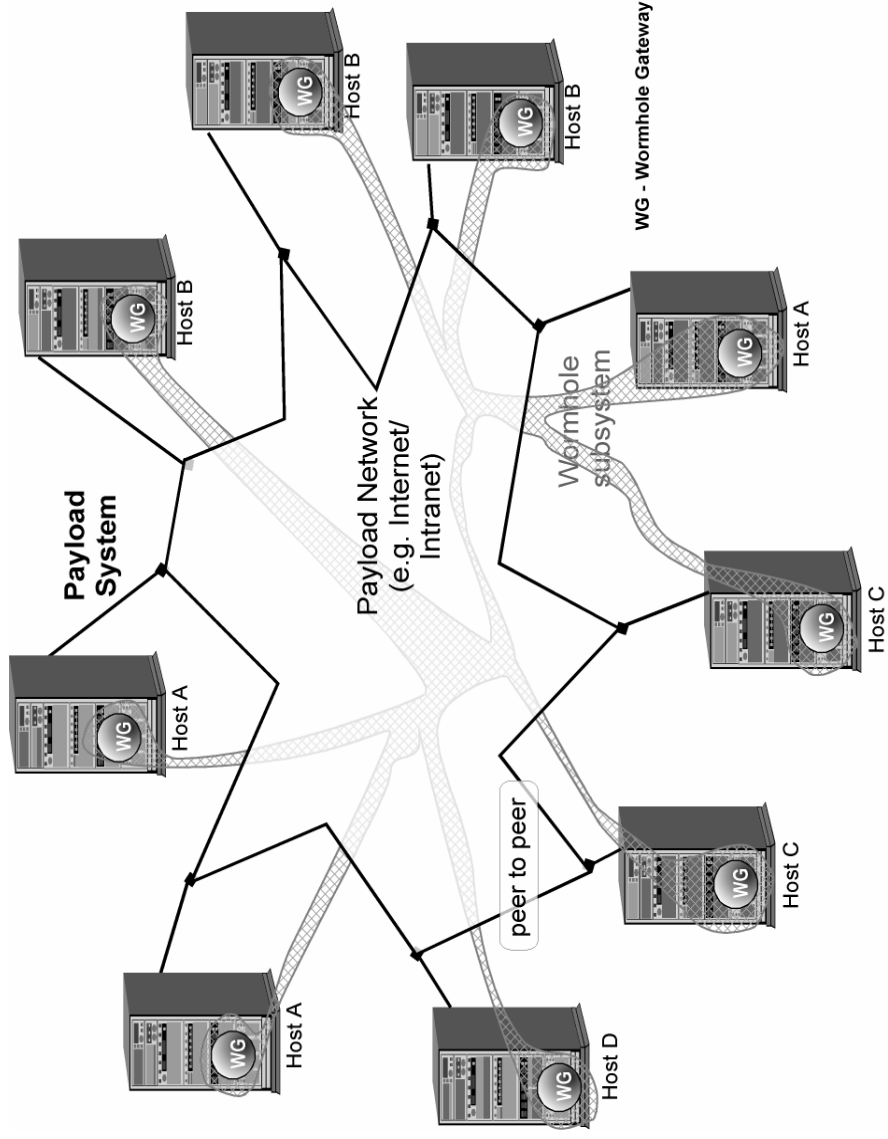


(a)

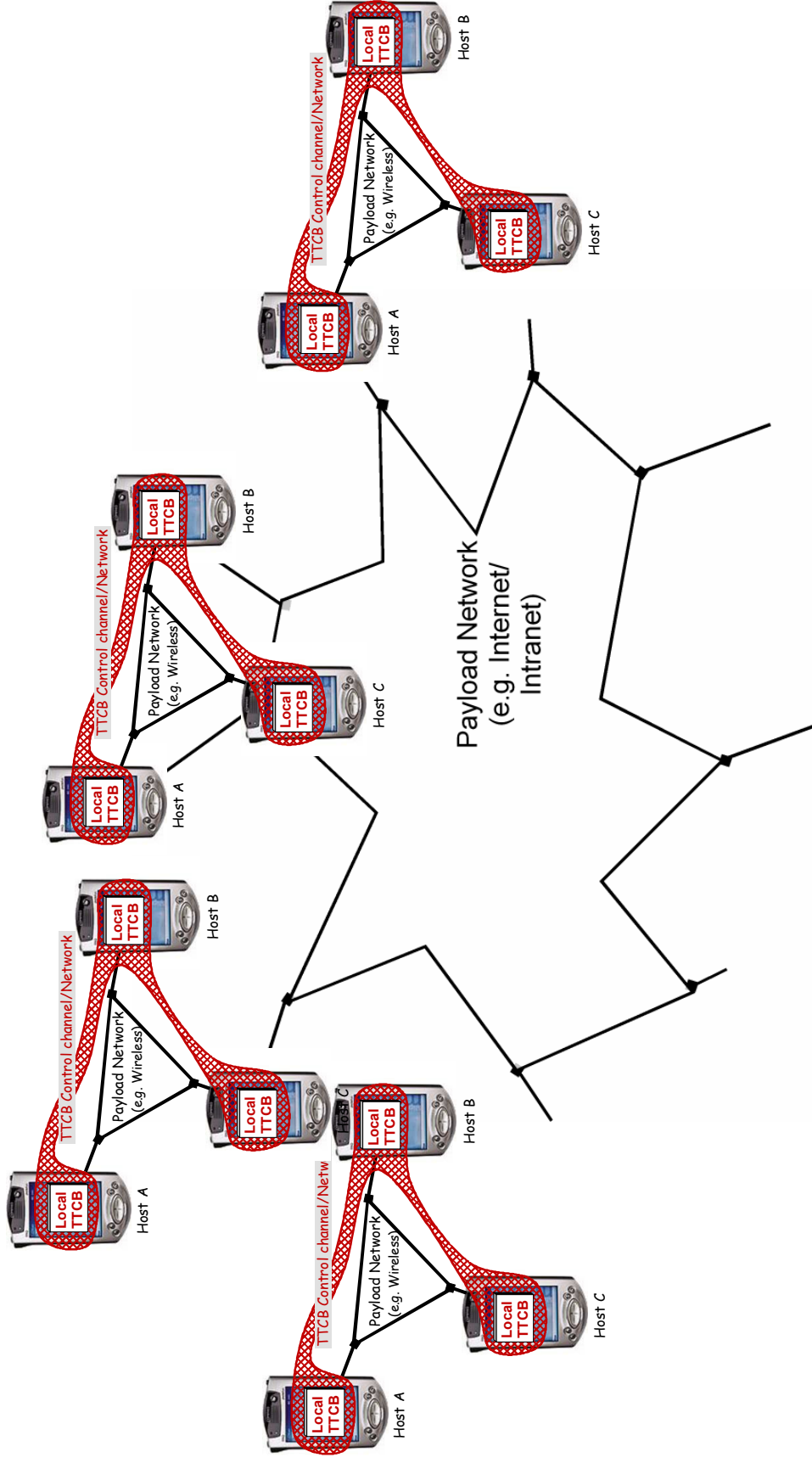


(b)

# Example of deployment of systems with wormholes



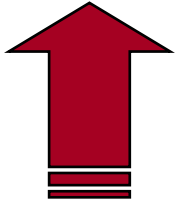
# Example of deployment of systems with wormholes



# A Concrete Model

## (the timeliness wormhole example)

### Timely Computing Base



# TCB Model

## TCB Services and Interface

## Implementation of a TCB

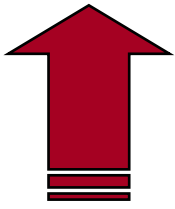
# Quest for a Generic Solution

- How to encompass the entire spectrum of *synchrony*, from fully sync to fully *async*?
- How to enforce certain time-related properties rather than waiting for them to happen?
- We devised a model that encompasses the entire spectrum of *partial synchrony*
  - 👉 Timely Computing Base (TCB) Model
- We devised an architecture that enforces timeliness
  - 👉 Timely Computing Base (TCB) Architecture

# A timeliness wormhole

- **Timely Computing Base wormhole:**
  - ☞ postulates space-domain heterogeneity (vis-a-vis synchronism)
    - components with “better” properties
  - ☞ monitors time-domain heterogeneity (synchrony variations)
    - payload can have any synchronism
  - ☞ offers simple SYNC services
    - TE - Timely execution, DM - Duration measurement, TFD - Timing failure detection
- **Timely Computing Base architecture :**
  - ☞ enforces architectural hybridization
    - construction of the “better” components
  - ☞ enforces sync/async interface
    - payload enjoying “better” components services

# TCB Model



# TCB Services and Interface

# Implementation of a TCB

# Services and API of the TCB

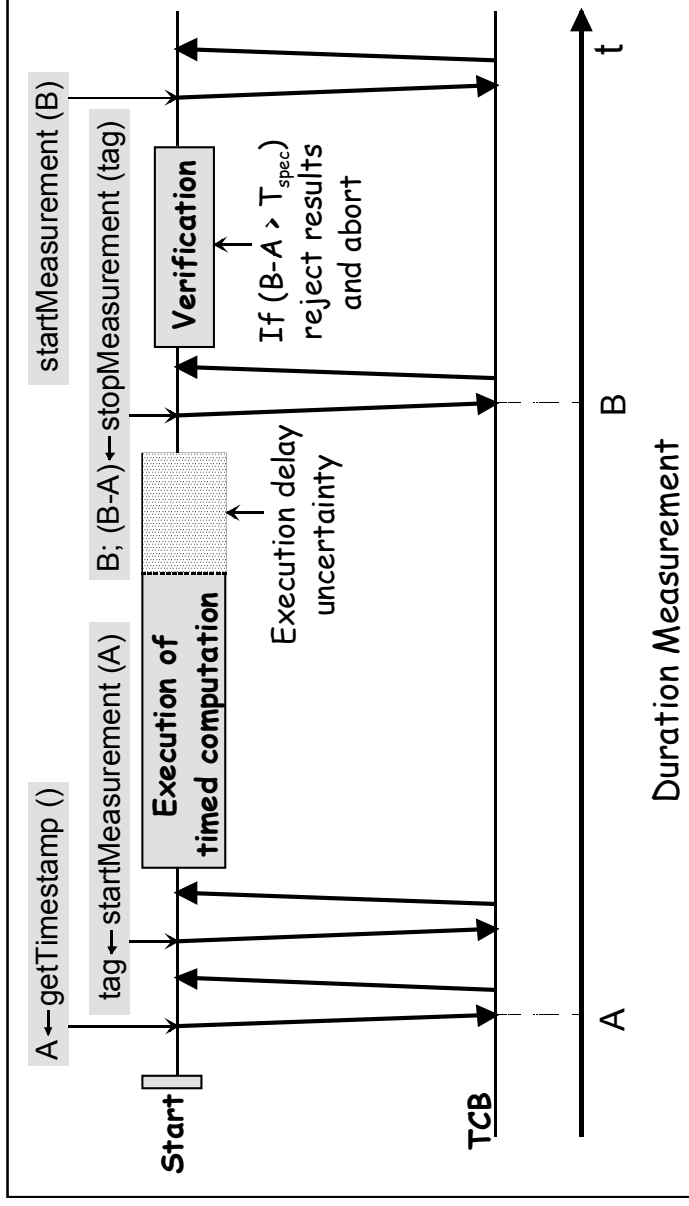
- The TCB provides minimal services:
  - ☞ TE - Timely execution
  - ☞ DM - Duration measurement
  - ☞ TFD - Timing failure detection
- And a payload-to-TCB interface
  - ☞ Allows potentially asynchronous applications to dialogue with a synchronous component

# Synch/Asynch Interface

- Important issues to retain:
  - ☞ The TCB does not make applications timelier
  - ☞ Service invocation latency not bounded
  - ☞ Service responses or timing failure notifications not bounded
  - ☞ Nothing obliges applications to become aware of failures
- The TCB as an oracle:
  - ☞ Applications take advantage of the TCB by construction
  - ☞ They observe correctness of past steps before proceeding
  - ☞ Timeliness always observed in terms of **durations**
  - ☞ Time-critical responses to failures handled by the TCB

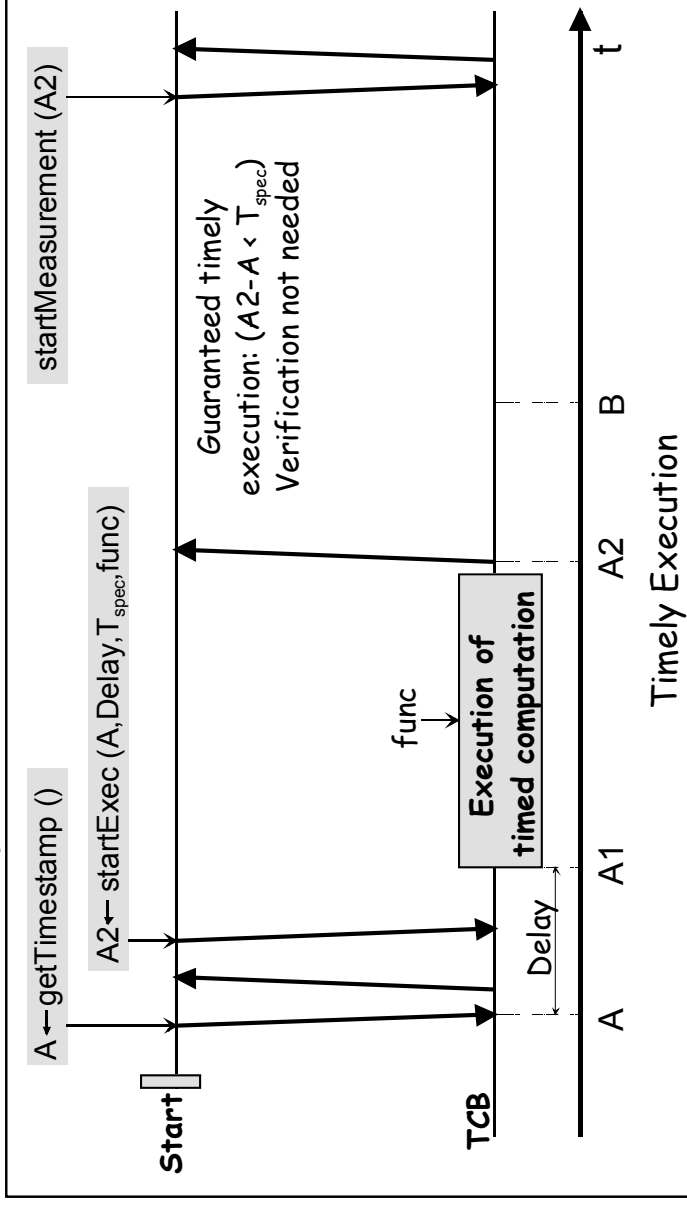
# API

- Duration Measurement
  - ☞ Allows the measurement of upper bounds of payload actions



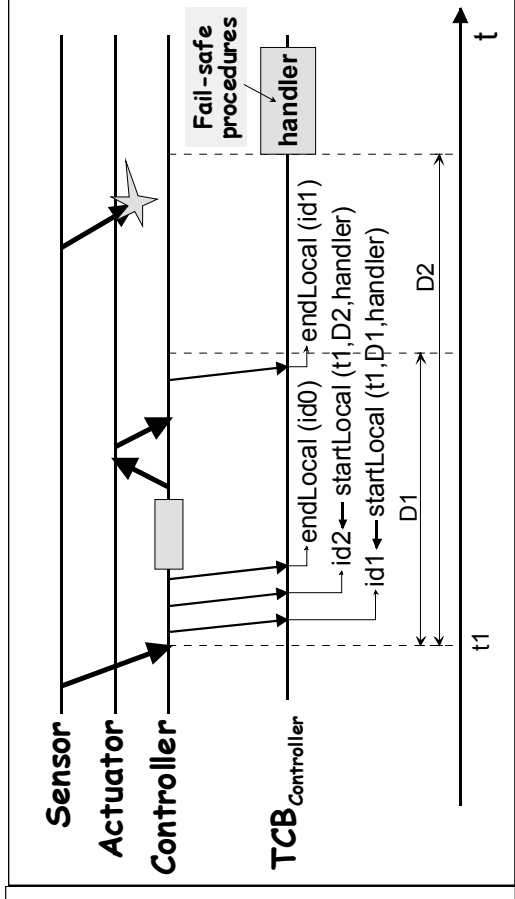
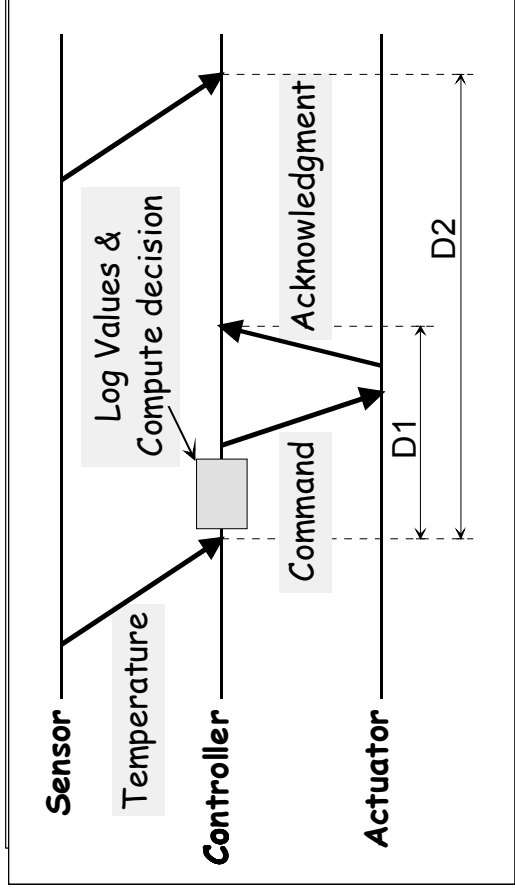
# API

- Timely Execution
  - ↳ Allows the timely execution of small time-critical functions

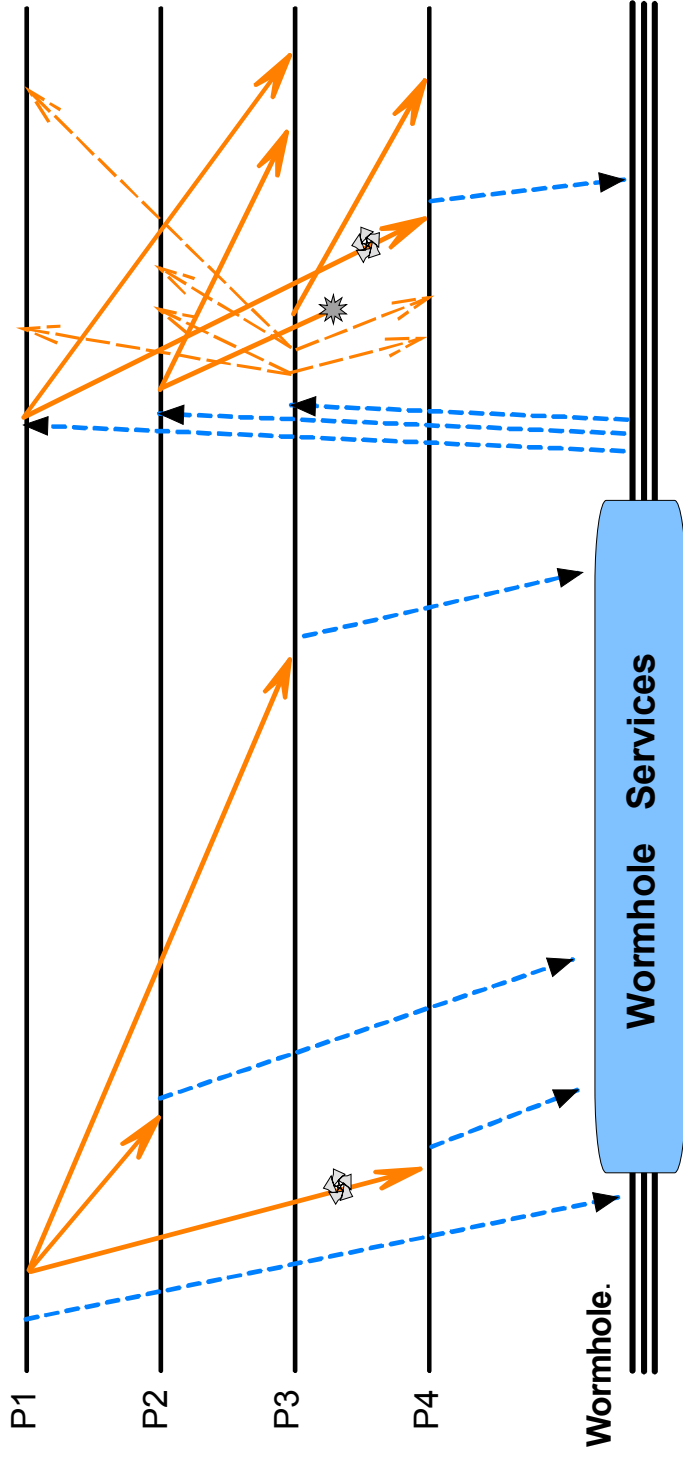


# API

- Timing Failure Detection
  - ☞ Allows detection of timing failures (of local and distributed actions)
  - ☞ Allows **timely execution of safety procedures** upon the detection of a timing failure

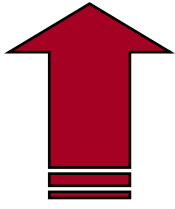


# Rationale of the operation of Wormhole-aware protocols



# TCB Model

## TCB Services and Interface

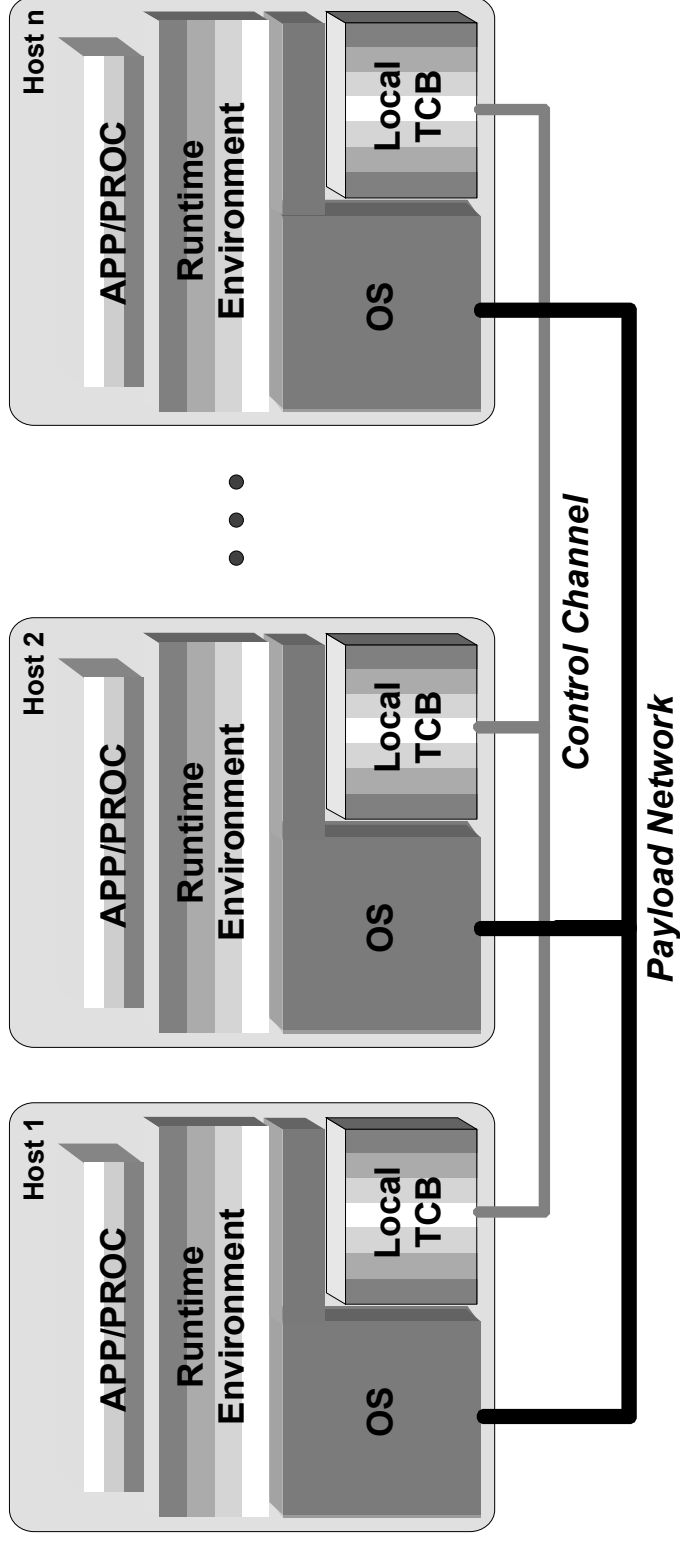


## Implementation of a TCB




# Implementation Issues

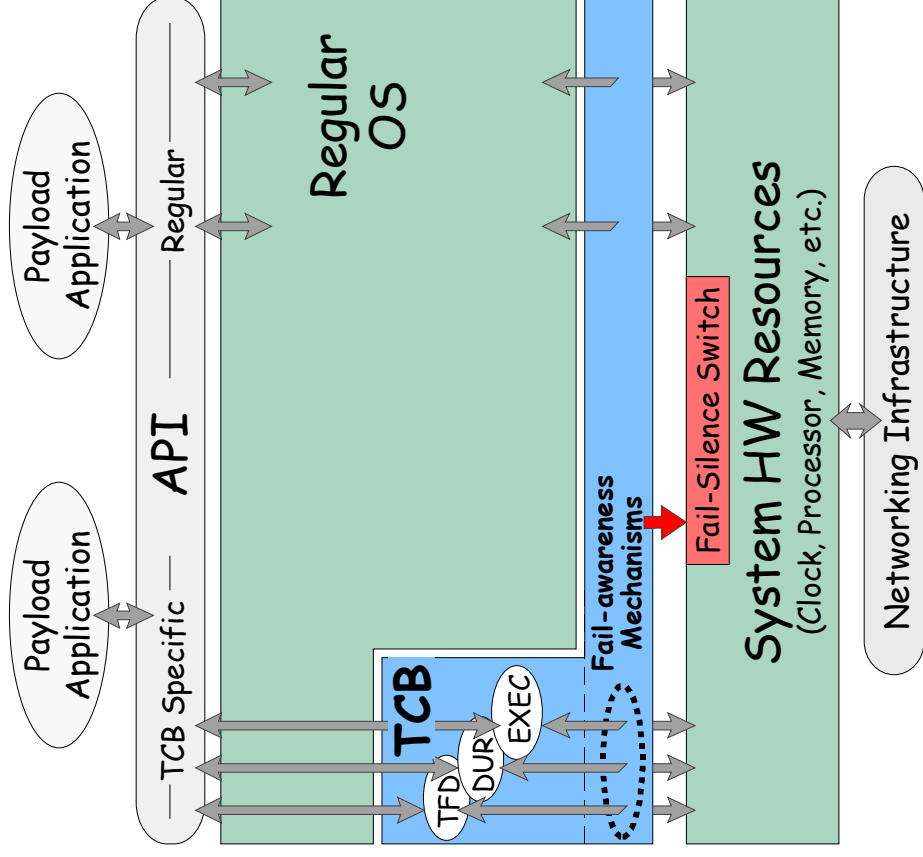
- Duration measurement
  - ☞ Local durations: local clock
  - ☞ Distributed durations: round-trip duration measurement technique
- Timely execution
  - ☞ Small functions residing in the TCB address space
  - ☞ Use known techniques of real-time systems
    - Admission control, schedulability analysis, WCET calculation
- Timing failure detection
  - ☞ Distributed protocol to ensure **(Timed) Completeness and Accuracy**
  - ☞ Set up timeouts using timers (based on local clock)

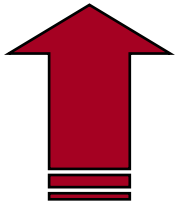
# A concrete example



# A System with a TCB

- 
 The TCB is built with given **<bound, coverage>** pairs
- 
 Hardware and inter-TCB communication channel may assume different forms, for different **<bound, coverage>** pairs
- 
 Software kernel on a plain desktop (PC or workstation) may be used





## **TCB Model (cont.)**

## **TCB Services and Interface**

## **Implementation of a TCB**

# What is the philosophy? (time domain example)

- Timing failures more complex than they look
  - ☞ **Unexpected delay** - "normal" effect
  - ☞ **Contamination** (of safety props) - error propagation effect
  - ☞ **Decreased coverage** - continued (statistical) effect
- Can we achieve correct operation despite these?
  - ☞ Contamination should be avoided at all cost
  - ☞ Coverage should remain stable

# Why is the framework generic

- Generic properties dictate correctness of applications, regardless of functional semantics
  - ☞ **Coverage Stability** - coverage of timing assumptions remains stable
  - ☞ **No-Contamination** - safety properties not violated
- Under uncertain timeliness, different classes of appl's secure these properties in different ways
- It is necessary to detect timing failures, and react to that

**TIMING FAILURE DETECTOR (TFD) considered fundamental**

# Dependable and Adaptive Computing with a TCB wormhole

- Introduce classes of applications that deal with these problems when assisted by a TCB:
  - ☞ **Fail-safe**: exhibits correct behaviour or stops in fail-safe state
  - ☞ **Time-elastic**: exhibits coverage stability
  - ☞ **Time-safe**: exhibits no-contamination
- Apply known fault tolerance techniques to the application classes (or combinations thereof):
  - ☞ **detection and/or recovery; masking**
- *The TCB Model and Architecture [ieeeTCS2002]*

# Example application frameworks

- **Fail-safe operation [DSN2000]** :
  - by switching to a fail-safe state after the first failure
  - requires the TFD service and appl's to be of the fail-safe class
- **Reconfiguration and adaptation [SRDS2001]** :
  - by enforcing coverage stability
  - requires appl's to be of the time-elastic and time-safe class
- **Timing error masking [DSN2002]**:
  - by using replication to mask transient timing errors
  - requires the TFD service and appl's to be time-safe class

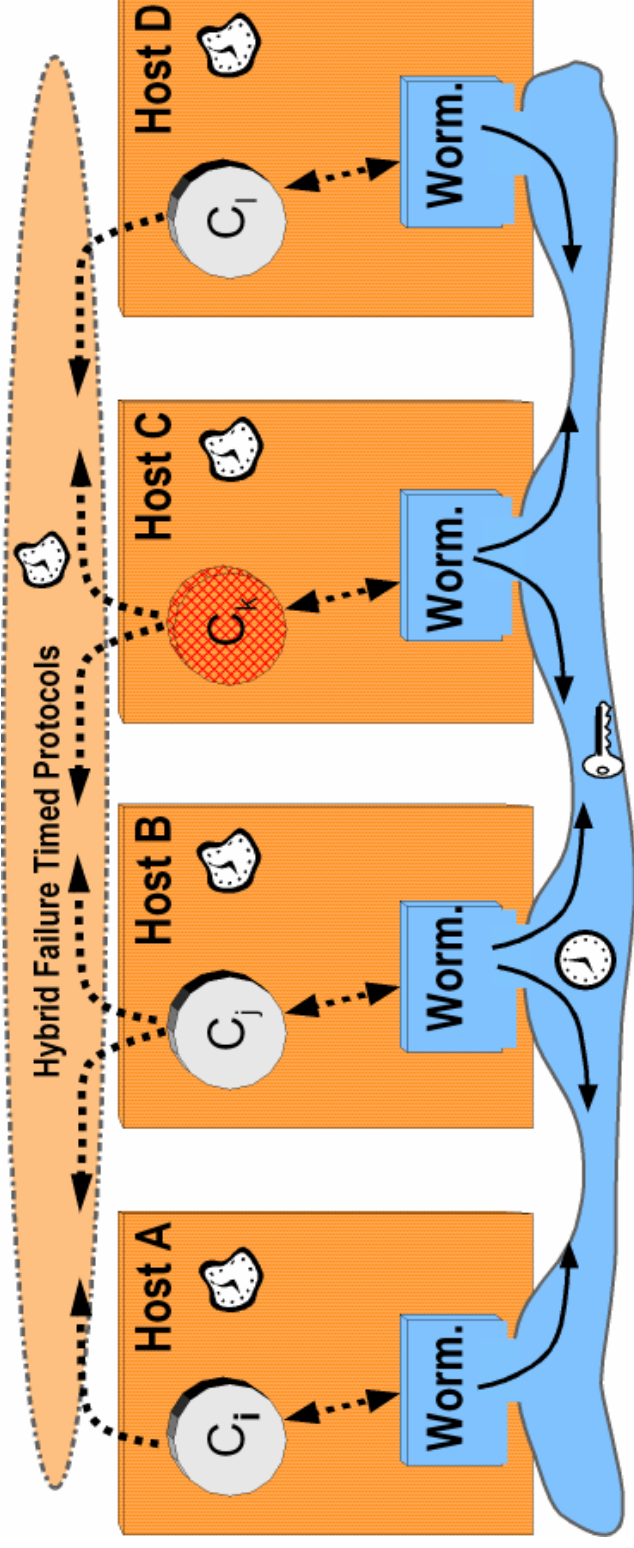
# Extending the wormhole concept to other domains

- The Wormholes model is generic: the “better” properties secured by the wormholes may be other than synchrony/timeliness

# Security Wormholes

- **TCB** - *Goal*: assisting potentially asynchronous applications/protocols through a synchronous device.  
*Problems*: the interface between the asynchronous payload and synchronous subsystem; implementing the synchronous subsystem
- **TrustedTCB** - *Goal*: assisting potentially insecure applications/protocols through a secure device.  
*Problems*: the interface between the insecure payload and secure subsystem; implementing the secure subsystem [EDCC2002]

# Strategy for security awareness and/or assurance



- - Fully secure (tamperproof), timely
- - Malicious, potentially untimely

# Trusted Timely Computing Base (TTCB)

- **Properties:**
  - trusted and timely execution assistant, a real-time distributed security-kernel with a limited set of services
- **Assists the execution of malicious FT algorithms:**
  - ☞ provides a fail-controlled environment that substantiates the hybrid failure model
- **Trusted versions of TCB Time-related Services**
- **TTCB Security-related Services**
  - ☞ *Trusted block agreement*
  - ☞ *Local component authentication*
  - ☞ *Trusted random number generation*

# Conclusions

- Some achievements...
- (T)TCB wormhole prototypes
  - Software Available at <http://www.navigators.di.fc.ul.pt/software/tcb>
- Current work:
  - ☞ Timing fault tolerance for event-based systems
  - ☞ Byzantine-resilient reliable multicast [SRDS2002]
  - ☞ In preparation: Byzantine-resilient consensus, atomic multicast and membership
- See more at: [www.navigators.di.fc.ul.pt](http://www.navigators.di.fc.ul.pt) -- “Documents”

# Some Recent Publications (urls)

- **Modeling Wormholes**
- *Uncertainty and Predictability: Can they be reconciled?* **Paulo Veríssimo**. Future Directions in Distributed Computing, pages to appear, Springer-Verlag LNCS 2584, month to appear, 2003
- *The Timely Computing Base Model and Architecture*. **Paulo Veríssimo, António Casimiro**. IEEE Transactions on Computers - Special Issue on Asynchronous Real-Time Systems, vol. 51, n. 8, Aug 2002
- *The Timely Computing Base: Timely Actions in the Presence of Uncertain Timeliness*. **Paulo Veríssimo, António Casimiro, C. Fetzer**. In Proceedings of the 1st International Conference on Dependable Systems and Networks, New York, USA, June 2000.
- *The Timely Computing Base*. Paulo Veríssimo and António Casimiro. **Technical Report DI/FCUL TR 99-2, Department of Informatics, University of Lisboa**, May 1999. (original paper, improved in TOCS02)
- **Implementing Wormholes**
- *Measuring Distributed Durations with Stable Errors*. **António Casimiro, Pedro Martins, Paulo Veríssimo, Luís Rodrigues**. Proceedings of the 22nd IEEE Real-Time Sysy Symposium, London, UK, December 2001
- *How to Build a Timely Computing Base using Real-Time Linux*. **António Casimiro, Pedro Martins, Paulo Veríssimo**. in Proceedings of the 2000 IEEE International Workshop on Factory Communication Systems, Porto, Portugal, September 2000.
- *Timing Failure Detection with a Timely Computing Base*. **António Casimiro, Paulo Veríssimo**. 3rd Europ. Research Seminar on Advances in Distr. Sys (ERSADS'99), Madeira Island, Portugal, April 23-28, 1999
- *The Design of a COTS Real-Time Distributed Security Kernel*, **Miguel Correia, Paulo Veríssimo, Nuno Ferreira Neves, Fourth European Dep. Comp. Conf.**, Toulouse, France, October 2002 © Springer-Verlag.
- **Using Wormholes**
- *Using the Timely Computing Base for Dependable QoS Adaptation*. **António Casimiro, Paulo Veríssimo**. Proceedings of the 20th IEEE Symp. on Reliable Distributed Systems, New Orleans, USA, October 2001
- *Generic Timing Fault Tolerance using a Timely Computing Base*. **António Casimiro, Paulo Veríssimo**. Procs of the Intern'l Conference on Dependable Systems and Networks, Washington D.C., USA, June 2002
- *Efficient Byzantine-Resilient Reliable Multicast on a Hybrid Failure Model*, **Miguel Correia, Lau Cheuk Lung, Nuno Ferreira Neves, Paulo Veríssimo**. Proc's of the 21st Symp. on Reliable Distributed Systems (SRDS'2002), Suita, Japan, October 2002