High Level Design Proof of a Reliable Computing Platform

Ben L. Di Vito
Ricky W. Butler
James L. Caldwell

NASA Langley Research Center
Hampton, VA 23665

Outline

- Research Objectives
- Reliable Computing Platform
- High-Level Design Specifications
- Correctness Proofs
- Voting Patterns

Digital Flight Control Systems

Aerodynamic Properties

Continuous Differential Equations Model

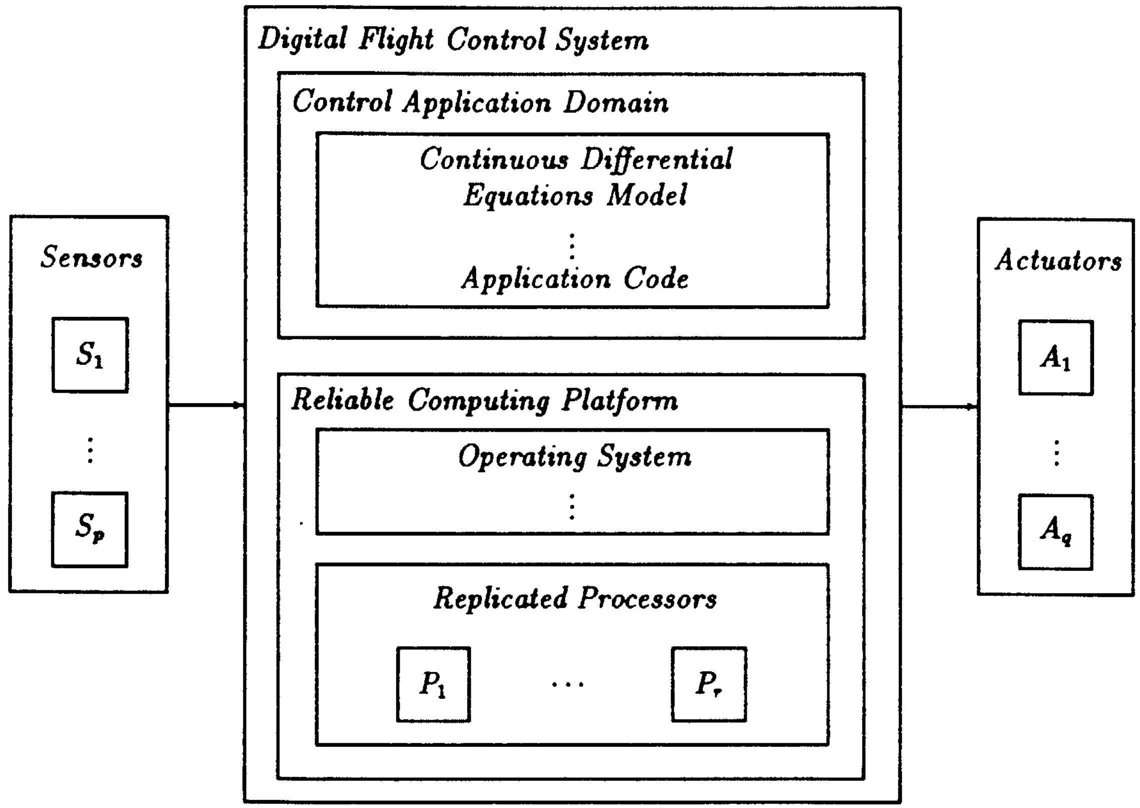
Control Law Block Diagram

Application Code

Operating System

Redundant Hardware

Reliable Computing Platform



Research Objectives

- Establish hardware/software platform for ultra-reliable computing
- Use fault-tolerant computer architecture
- Use formal methods to prevent design and implementation errors
  - first specify in conventional mathematical notation
  - then specify and mechanically verify in EHDM
- Construct reliability model to quantify reliability estimate

Operating System for Control Applications

Single Processor Abstraction

Fault-tolerant Synchronous Replicated Model

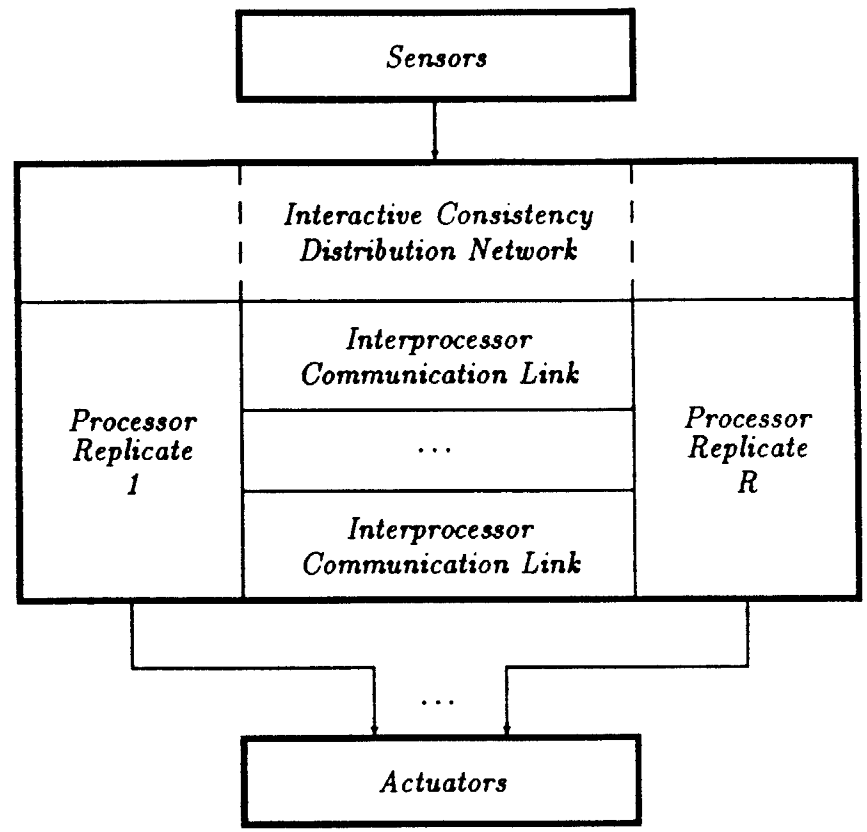
Fault-tolerant Asynchronous Replicated Model

Hardware/Software Implementation

Application Task Characteristics

1. Fixed set of tasks
2. Hard deadlines
3. Multi-rate cyclic scheduling
4. Minimal jitter
5. Upper bound on task execution time
6. Precedence constraints

Architectural Concept

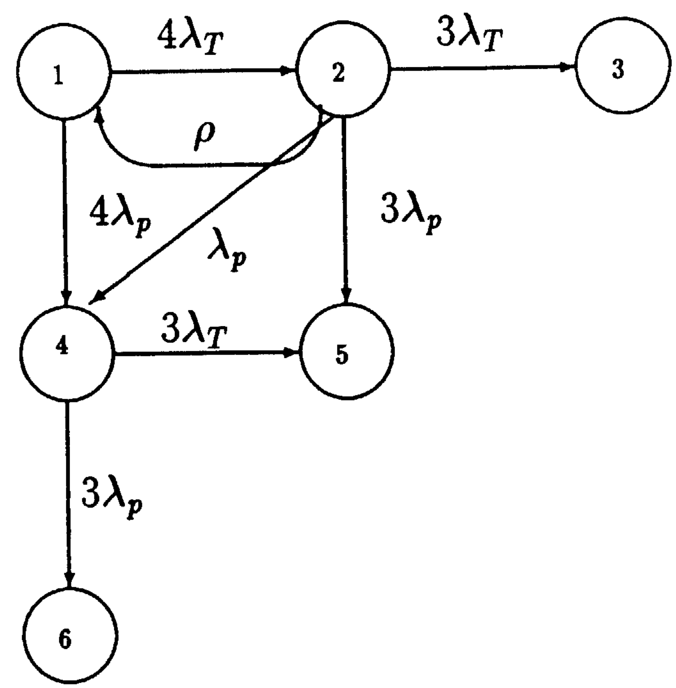


Design Decisions

1. the system is non-reconfigurable
2. the system is frame-synchronous
3. the scheduling is static, non-preemptive
4. internal voting is used which can recover the state of a processor affected by a transient fault within a bounded time

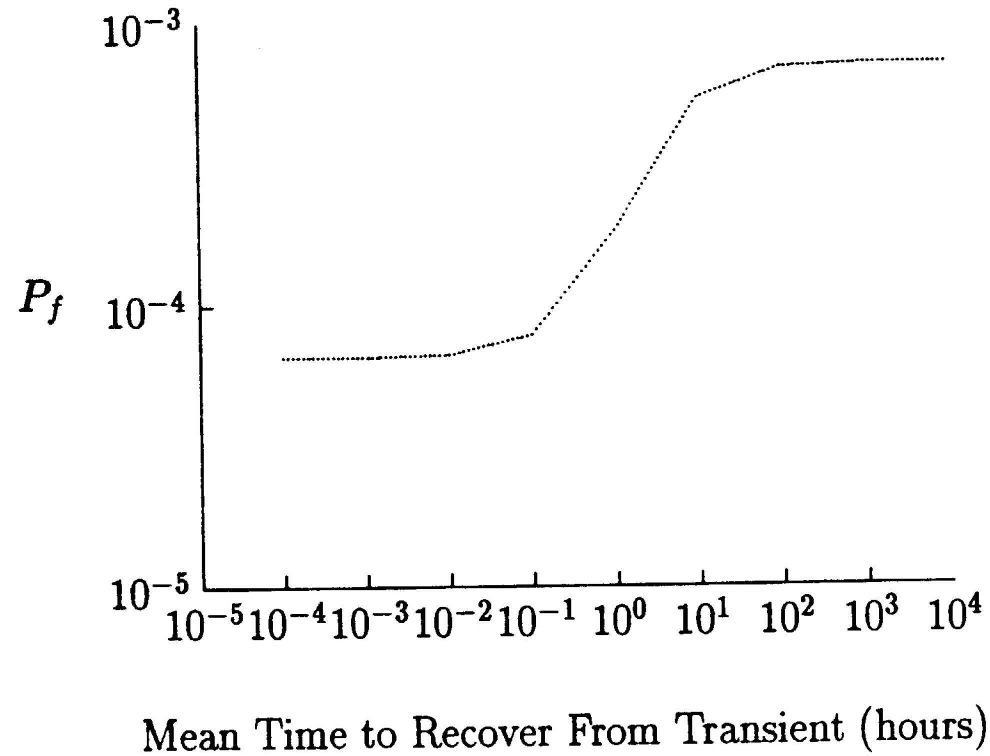
Reliability Modeling

Reliability model of quadruplex version of system



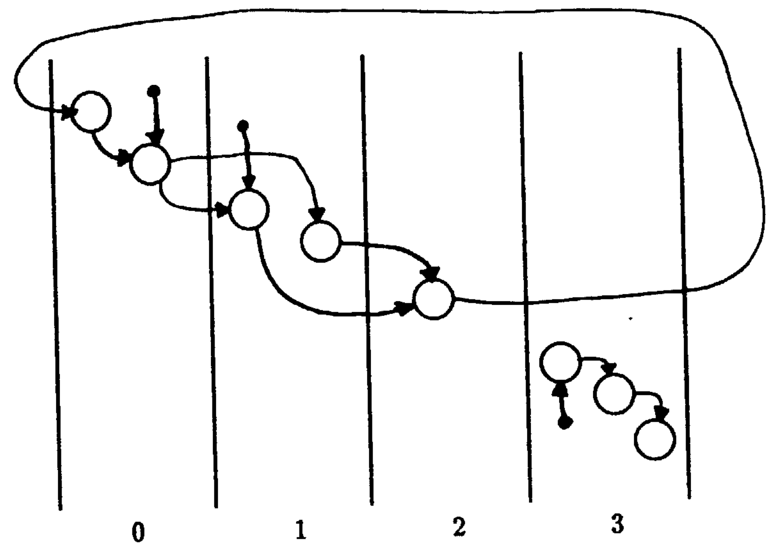
λ_T = transient fault rate ($\sim 10^{-3}$/hr)
λ_P = permanent fault rate ($\sim 10^{-4}$/hr)
ρ = rate of recovery from transient fault (design-dependent)

Transient Fault Recovery



Note inflection point on the order of one minute

Application Definition



M frames = 1 cycle

$M_i > 0$ subframes per frame

K tasks

(i, j) = cell (frame,subframe)

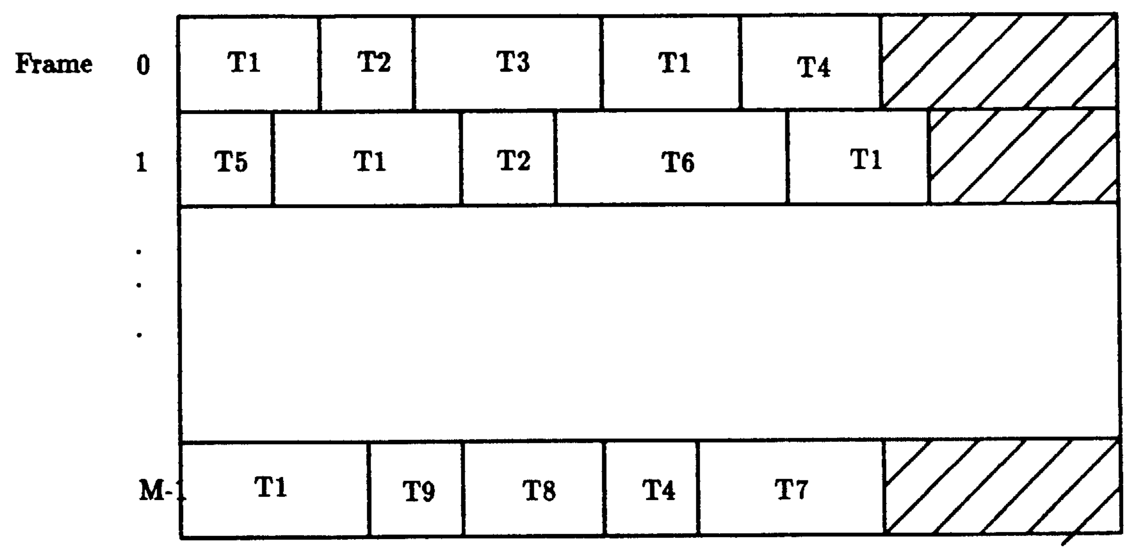
ST: scheduled task for cell (i, j)

TI: task inputs for cell (i, j) {tasks have no permanent state}

AO: actuator output tasks

IR: initial task inputs

Task Schedule



Uniprocessor Model

State of abstract machine given by:

$$\begin{aligned} OS\_state = (\ & frame : \{0..M-1\}, \\ & results : \{0..M-1\} \times nat \to D \ ) \end{aligned}$$

The OS state transition is defined by the function OS.

$$OS : Sin \times OS\_state \to OS\_state$$

$$OS(s,u) = (u.frame \oplus 1, \lambda i, j.\ new\_results(s,u,i,j))$$

where

$$x \oplus y = (x+y) \bmod M$$

$$x \ominus y = (x+M-y) \bmod M$$

$$\begin{aligned} new\_results(s,u,i,j) = \text{if } & i = u.frame \\ & \text{then } exec(s,u,i,j) \\ & \text{else } u.results(i,j) \end{aligned}$$

Uniprocessor Model (Cont'd.)

$$exec : Sin \times OS\_state \times \{0..M-1\} \times nat \rightarrow D$$

$$exec(s,u,i,j) = f_{ST(i,j)}(arg(TI(i,j)[1],s,u,i,j),\ldots,arg(TI(i,j)[n],s,u,i,j))$$

$$arg : triple \times Sin \times OS\_state \times \{0..M-1\} \times nat \rightarrow D$$

$$\begin{aligned} arg(t,s,u,i,j) = {} & \text{if } t.type = sensor \\ & \quad \text{then } s[t.i] \\ & \quad \text{else if } t.i = i \wedge t.j < j \\ & \quad\quad \text{then } exec(s,u,i,t.j) \\ & \quad\quad \text{else } u.results(t.i,t.j) \end{aligned}$$

Actuator output is a function of the OS state:

$$UA(u) = [\,\|_{k=1}^{q}\ Act(u,k)\,]$$

$$Act(u,k) = \begin{cases} u.results(u.frame \ominus 1, j) & \\ & \text{if } \exists j : AO(u.frame \ominus 1, j) = k \\ \phi & \text{otherwise} \end{cases}$$

Replicated Processor Model

The replicated processor model is based on a replicated state and transitions that allow for faults in the replicates

$$Repl : ICin \times Repl\_state \times fault\_status \rightarrow Repl\_state$$

$$Repl(c, r, \Phi) = [_{k=1}^{R} \; RT(c, r, k, \Phi) \,]$$

$$RT(c, r, k, \Phi) = \text{if } \Phi[k] \text{ then } \perp \text{ else } (frame\_vote(r, \Phi), Repl\_results(c, r, k, \Phi))$$

$$frame\_vote(r, \Phi) = maj([_{l=1}^{R} \; FV_l \,])$$

$$\text{where } FV_l = \text{if } \Phi[l] \text{ then } \perp \text{ else } r[l].frame \oplus 1$$

$$maj : sequence(D \cup \{\perp\}) \rightarrow D \cup \{\perp\}$$

Replicated Processor Model (Cont'd.)

$$VP : \{0..M-1\} \times nat \times \{0..M-1\} \rightarrow \{T, F\}$$

$VP(i, j, n) = T$ iff we are to vote $OS.results(i, j)$ during frame n.

$$\begin{aligned} &Repl\_results(c, r, k, \Phi) = \\ &\qquad \lambda i, j.\ \text{if } VP(i, j, r[k].frame) \\ &\qquad\qquad \text{then } results\_vote(c, r, i, j, \Phi) \\ &\qquad\qquad \text{else } new\_results(c[k], r[k], i, j) \end{aligned}$$

$$results\_vote(c, r, i, j, \Phi) = maj([_{l=1}^{R}\ RV_l\ ])$$

where $RV_l = \text{if } \Phi[l] \text{ then } \perp \text{ else } new\_results(c[l], r[l], i, j)$.

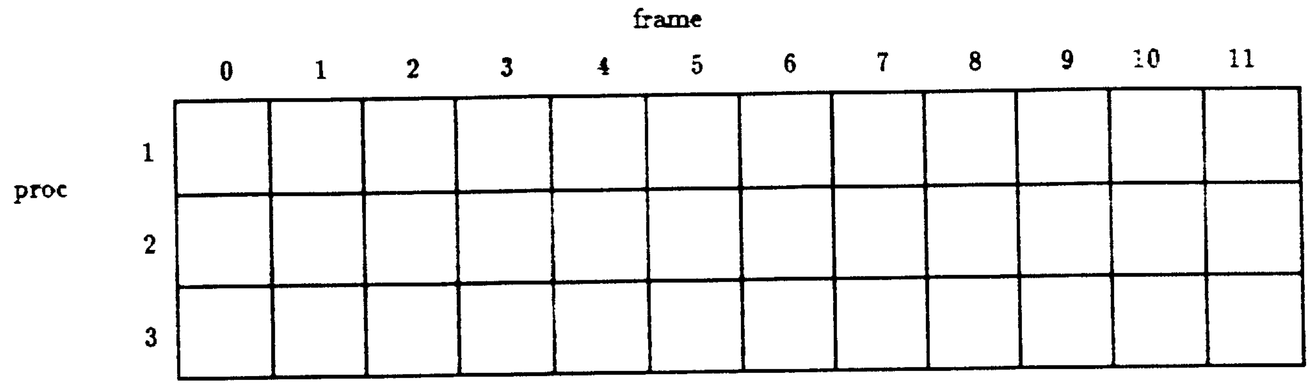
Replicated actuator output considers fault status indicators:

$$RA : Repl\_state \times fault\_status \rightarrow RAout$$

$$RA(r, \Phi) = [_{k=1}^{R}\ RA_k\ ]$$

where $RA_k = \text{if } \Phi[k] \text{ then } \perp \text{ else } UA(r[k])$

A Simple Fault Model



The results we seek must hold for all $\mathcal{F} : \{1..R\} \times nat \rightarrow \{T, F\}$ that satisfy a condition for maximally unfortunate fault behavior. Define a *working* processor as follows.

$$\mathcal{W} : \{1..R\} \times nat \times fault\_fn \rightarrow \{T, F\}$$

$$\mathcal{W}(k, n, \mathcal{F}) = \forall j : \ 0 \leq j \leq min(n, N_R) \supset \sim \mathcal{F}(k, n - j)$$

A processor that is nonfaulty, but not yet working, is considered to be *recovering*. The number of working processors is given by:

$$\omega(n, \mathcal{F}) = |\{k \mid \mathcal{W}(k, n, \mathcal{F})\}|$$

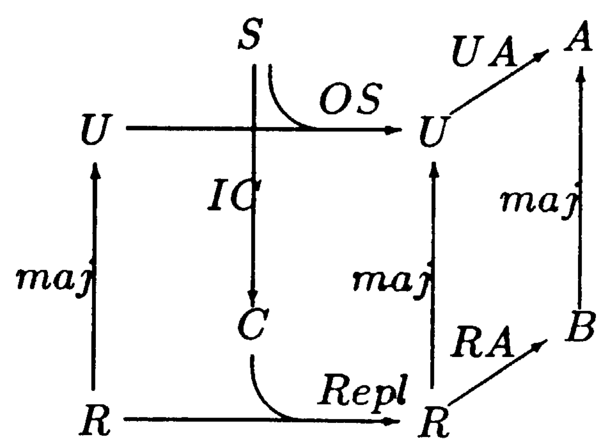
Definition 1 *The* Maximum Fault Assumption *for a given fault function $\mathcal{F}$ is that $\omega(n, \mathcal{F}) > R/2$ for every frame n.*

All theorems about state machine correctness are predicated on this assumption.

Framework For Proving State Machine Correctness

Functions needed to bridge the gap between the two machines are those that do the following:

1. Map sensor inputs for UM into replicated sensor inputs for RM.
2. Map replicated actuator outputs from RM into actuator outputs for UM.
3. Map replicated OS states of RM into uniprocessor OS states of UM.



Correctness Criteria

Definition 2 *RM correctly implements UM under assumption $\mathcal{P}$ iff the following formula holds:*

$$\forall \mathcal{F} : \mathcal{P}(\mathcal{F}) \supset \forall S.\ \forall n > 0 : \ a_n = \nu(b_n) \tag{1}$$

where a_n and b_n can be characterized as functions of an initial state and all prior inputs.

We parameterize the concept of necessary assumptions using the predicate $\mathcal{P}$. For the replicated system, it will be instantiated by the Maximum Fault Assumption:

$$\mathcal{P}(\mathcal{F}) = (\forall m : \ \omega(m, \mathcal{F}) > R/2).$$

Derived Correctness Criteria

Definition 3 (Replicated OS Correctness Criteria) RM correctly implements UM *if the following conditions hold:*

$$(1)\quad u_0 = maj(r_0)$$

$$(2)\quad \forall \mathcal{F}.\ (\forall m:\ \omega(m, \mathcal{F}) > R/2) \supset \\ \forall S.\ \forall n > 0:\ OS(s_n, maj(r_{n-1})) = maj(Repl(IC(s_n), r_{n-1}, \mathcal{F}_n^R))$$

$$(3)\quad \forall \mathcal{F}.\ (\forall m:\ \omega(m, \mathcal{F}) > R/2) \supset \\ \forall S.\ \forall n > 0:\ UA(maj(r_n)) = maj(RA(r_n, \mathcal{F}_n^R))$$

Sufficient Conditions for Correctness

Generic State Machine Correctness Criteria

$\Uparrow$

Replicated OS Correctness Criteria

$\Uparrow$

Consensus Property

$\Uparrow$

Replicated State Invariant

$\Uparrow$

Full Recovery Property

$\Uparrow$

Voting Pattern

Intermediate Assertions

Definition 4 (Consensus Property) *For $\mathcal{F}$ satisfying the Maximum Fault Assumption, the assertion*

$$\mathcal{W}(p, n-1, \mathcal{F}) \supset r_{n-1}[p] = maj(r_{n-1}) \wedge r_n[p] = maj(r_n)$$

holds for all p and all $n > 0$.

Definition 5 (Replicated State Invariant) *For fault function $\mathcal{F}$ satisfying the Maximum Fault Assumption, the following assertion is true for every frame n:*

$$\begin{array}{l}
(n = 0 \vee \sim \mathcal{F}(p, n-1)) \supset \\
\quad r_n[p].frame = maj(r_n).frame = n \bmod M \wedge \\
\quad (\forall i, j :\ rec(i, j, \mathcal{L}(p, n, \mathcal{F}), \mathcal{H}(p, n, \mathcal{F}), T) \supset \\
\qquad r_n[p].results(i, j) = maj(r_n).results(i, j)).
\end{array}$$

Recovery Concepts

Recovery of state element (i, j) where last faulty frame was f and processor has been healthy for h frames:

$$
\begin{aligned}
rec(i,j,f,h,e) = {} & \text{if } h \le 1 \text{ then } F \\
& \text{else } (VP(i,j,f \oplus h) \wedge e) \vee \\
& \quad \text{if } i = f \oplus h \\
& \quad\quad \text{then } \wedge_{l=1}^{|TI(i,j)|} RI(TI(i,j)[l], i, j, f, h) \\
& \quad\quad \text{else } rec(i,j,f,h-1,T)
\end{aligned}
$$

$$
\begin{aligned}
RI(t,i,j,f,h) = {} & (t.type = sensor) \vee \\
& \text{if } t.i = f \oplus h \wedge t.j < j \\
& \quad \text{then } rec(t.i, t.j, f, h, F) \\
& \quad \text{else } rec(t.i, t.j, f, h-1, T)
\end{aligned}
$$

Definition 6 (Full Recovery Property) *The predicate* $rec(i,j,f,N_R,T)$ *holds for all* i, j, f.

Continuous Voting

$VP(i, j, k) = T \quad \forall i, j, k$

$N_R = 2 \qquad \text{(Actual } N_R = 1\text{)}$

- Specifies that the *entire state* will be voted every frame
- Not very practical
- But proof is simple

Cyclic Voting

$VP(i, j, k) = (i = k) \quad \forall i, j, k$

$N_R = M + 1.$

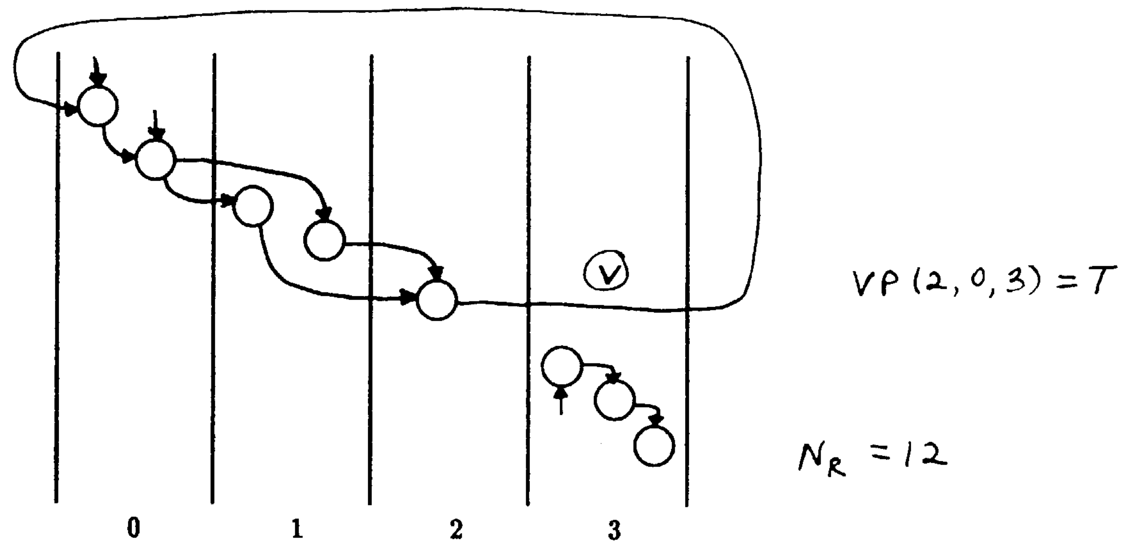
- Only results just computed will be voted in a frame
- More practical
- Proof almost as simple

Frame

Portion voted

0	1	2	3	0	1	2	3
v				v			
	v				v		
		v				v	
			v				v

Minimal Voting



- Vote only portion of state that will not be recovered from new sensor values.
- Construct VP to ensure each cycle of graph is cut by at least one vote.
- $N_R = L_C + L_N + M$ where
  - L_C = maximum frame length for all cycles
  - L_N = maximum frame length for all noncyclic paths

Summary

- Ultra-reliable control systems hard to achieve
- Simple fault-tolerant design postulated
- Formal specification of design constructed
- Preliminary correctness proofs obtained
- Will extend from here
  - more sophisticated designs
  - mechanical verification