Teste de Software Concorrente: quem precisa?

Simone Senger de Souza
Departamento de Sistemas de Computação
ICMC / USP
srocio@icmc.usp.br

SAST 2017 – 2º Simpósio Brasileiro de Teste de Software Sistemático e Automatizado – VIII CBSOFT – Fortaleza
Setembro/2017

Temas

- Teste de Software em diferentes domínios
  - Programação Concorrente

Apresentação Pessoal

- Início do meu relacionamento com **teste de software**
  - Avaliação do Custo e Eficácia do Critério Análise de Mutantes na Atividade de Teste de Programas

    (Mestrado 1996)
  - Validação de Especificações de Sistemas Reativos: Definição e Análise de Critérios de Teste

    (Doutorado 2000)
- Início do meu relacionamento com **teste de software concorrente**
  - Projeto CNPq PDPG-TI (2003): ValiPVM (Definição e Implementação de uma Ferramenta para Validação de Softwares Paralelos em Ambiente de Passagem de Mensagens)

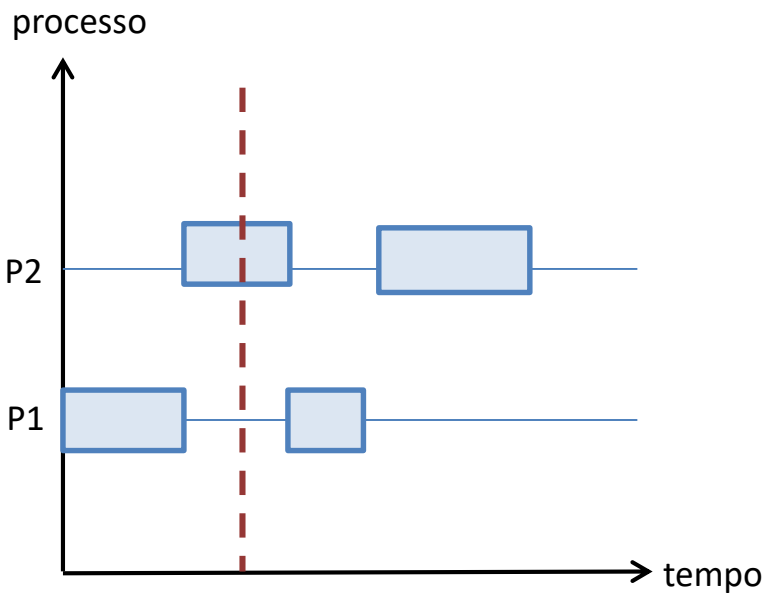
Definição Básica

Um **programa concorrente** contém dois ou mais processos ou *threads* que trabalham juntos para realizar uma tarefa.

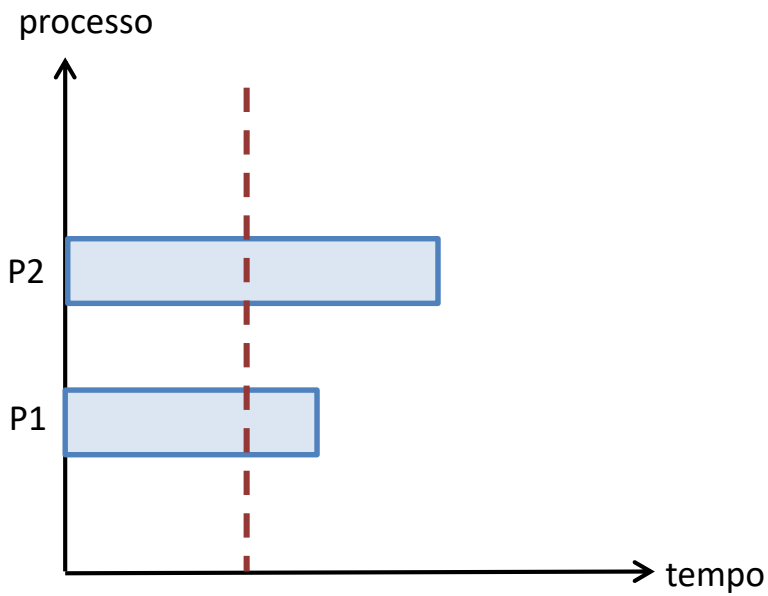
processos concorrentes = começaram a sua execução e em um determinado instante do tempo e ainda não a finalizaram

processos paralelos = executam em diferentes processadores no mesmo intervalo de tempo.

Programação Concorrente



processos concorrentes



processos paralelos

Programação Concorrente

- Paradigmas:
  - Memória compartilhada:
    - threads compartilham espaço de endereçamento (**variáveis compartilhadas**)
    - Sincronização através de monitores e semáforos.
  - Passagem de mensagens:
    - Processos possuem memória distribuída
    - Sincronização por troca de mensagens:
      - Primitivas **send/receive**

Teste de Software Concorrente: quem precisa disso?

Teste de Software Concorrente: quem não precisa disso?

Concorrência nos softwares modernos

- Reduzir o tempo de execução em diferentes domínios de aplicação:
  - Simulação de moléculas
  - Bioinformática
  - Processamento de imagens
  - Previsão do tempo
  - Novas drogas para o câncer
  - Origem do universo

Concorrência nos softwares modernos

- Re
  - P
  - o do tempo
  - ovas drogas para o câncer
  - Origem do universo

Se você não trabalha com esse domínio de aplicações, então você não precisa disso!

Super Computadores - HPC

Concorrência nos softwares modernos

- Aumentar taxa e garantir o atendimento:
  - Servidores Web
  - Sistemas embarcados (carro, aviões...)
  - Celulares

Uso de redes de computadores, clusters, processadores multinúcleos, cloud computing....

Concorrência nos softwares modernos

- Aumentar taxa e garantir o
  at

Se você trabalha com aplicações nesse domínio ou é **afetado** por ele, então seria bom se preocupar com testes!!!!

de redes de computadores, clusters, processadores multinúcleos, cloud computing....

Servidores Web

Aplicativos móveis

Automação industrial

Software embarcado em aviões

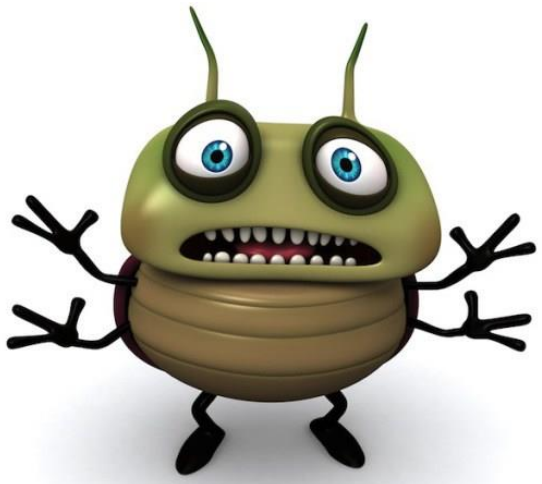
Software embarcado em carros

Software concorrente está presente no nosso dia a dia

Importante que esteja correto!

Falhas de Software mais famosas da História*

* *A Collection of Well-Known Software Failures* - http://www.cse.psu.edu/~gxt29/bug/softwarebug.html



Mars Pathfinder Mission

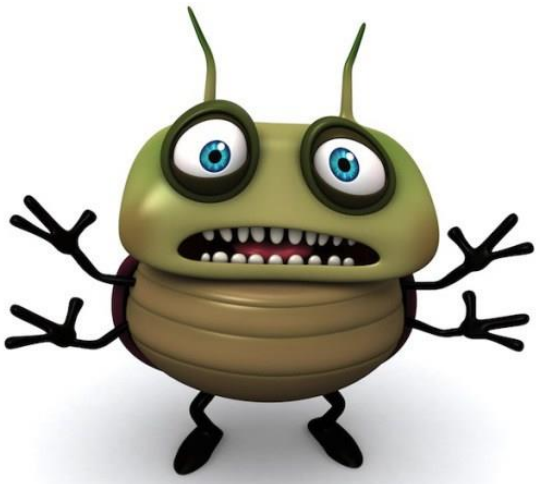
- Robô da NASA para explorar Marte (1997)
- Problema: Depois de pousar com sucesso tinha a missão de enviar dados sobre Marte para a Terra. Depois de alguns dias começou a reiniciar intermitentemente e parou de enviar os dados
- Causa: erro no módulo de comunicação entre processos: 3 processos em execução com diferentes prioridades (H, M, L): processo M começou a executar por longo tempo e processo H ficou esperando processo L finalizar. Porém processo L não conseguiu acesso à CPU porque M estava executando! Devido ao tempo sem evoluir, o dispositivo reiniciava o sistema, que voltava a mesma situação, devido ao bug.

Máquina de radiação Therac-25

- Evolução de modelos anteriores, a Therac-25 tinha os controles de segurança no software (1985)
- Problema: Devido à falha em mostrar o valor de radiação atribuído, pacientes receberam doses letais de radiação (3 mortes e 3 gravemente feridos)
- Causa: Problema conhecido como “condição de disputa”. O técnico tinha a impressão que o valor configurado estava correto, mas, devido ao bug, o valor era diferente do definido, aumentando os valores de radiação.

Por que essas falhas ainda ocorrem?

Podemos evitar essas falhas de software?



Programação Concorrente: Dificuldades

- Dificuldades para desenvolver programas concorrentes
  - Complexidade
  - Solução computacional x recursos computacionais
  - Propenso a erros (error prone)
  - Desenvolvedores não treinados
  - Testes não são realizados....

Programação Concorrente: Dificuldades

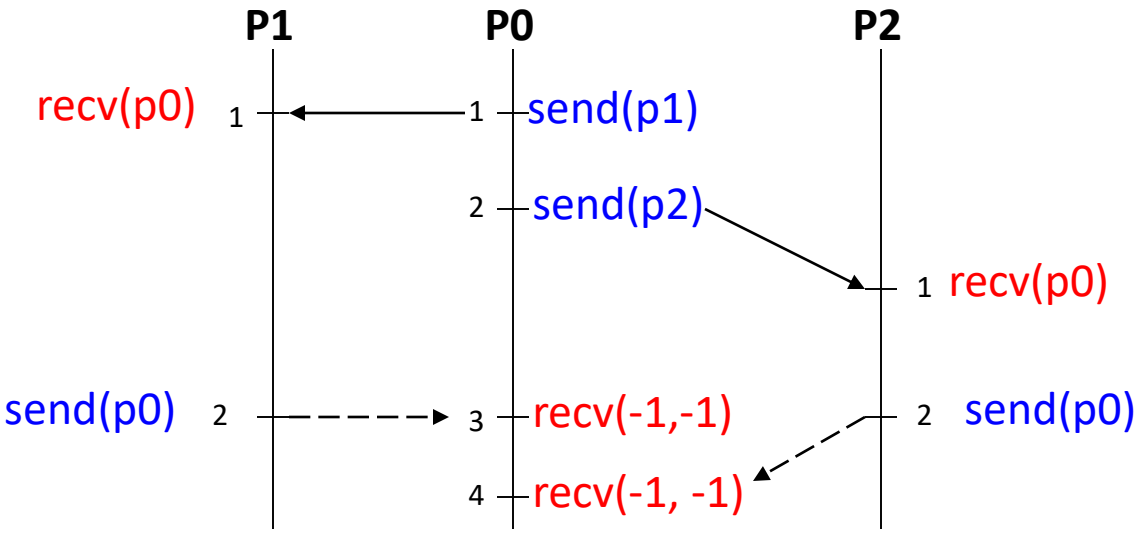
- Por que testes neste domínio são complexos?
  - Faltam ferramentas de teste
  - Reprodução de execuções prévias são difíceis
  - Novos desafios:
    - **Não determinismo**
    - Sincronização entre processos ou threads
    - Comunicação entre processos ou threads

Não determinismo

- Problema exemplo:
  - Considere um programa com três processos paralelos: P0 (mestre), P1 e P2
  - P0 inicia enviando dados para P1 e P2
  - P1 e P2 realizam suas computações e retornam os dados para P0
  - P0 recebe os dados de P1 e P2 de maneira não determinística (depende da velocidade de cada um)

Não determinismo

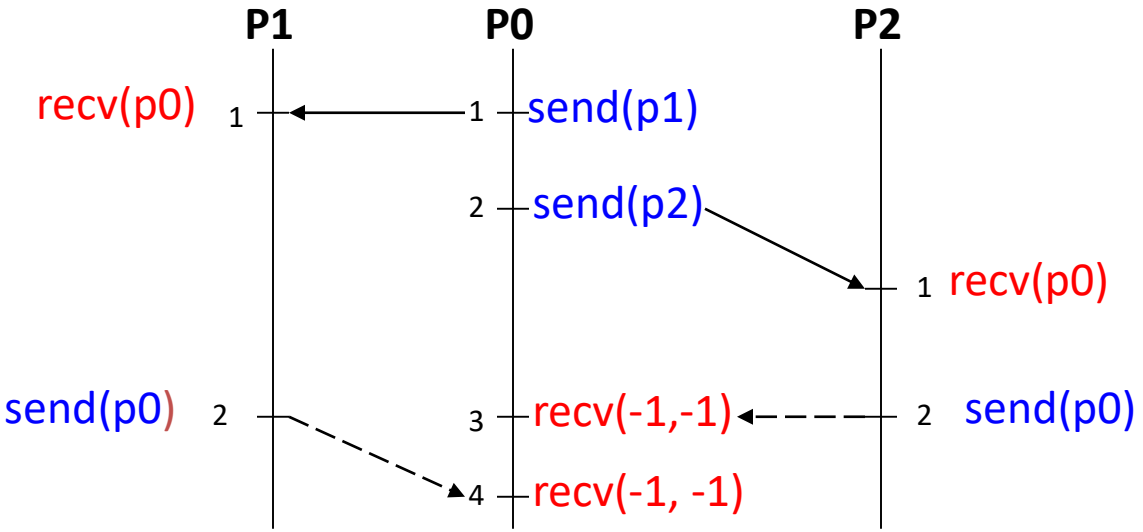
Timestamp 1



Sincronizações: $(1^0,1^1)$, $(2^0,1^2)$, $(2^1,3^0)$, $(2^2,4^0)$

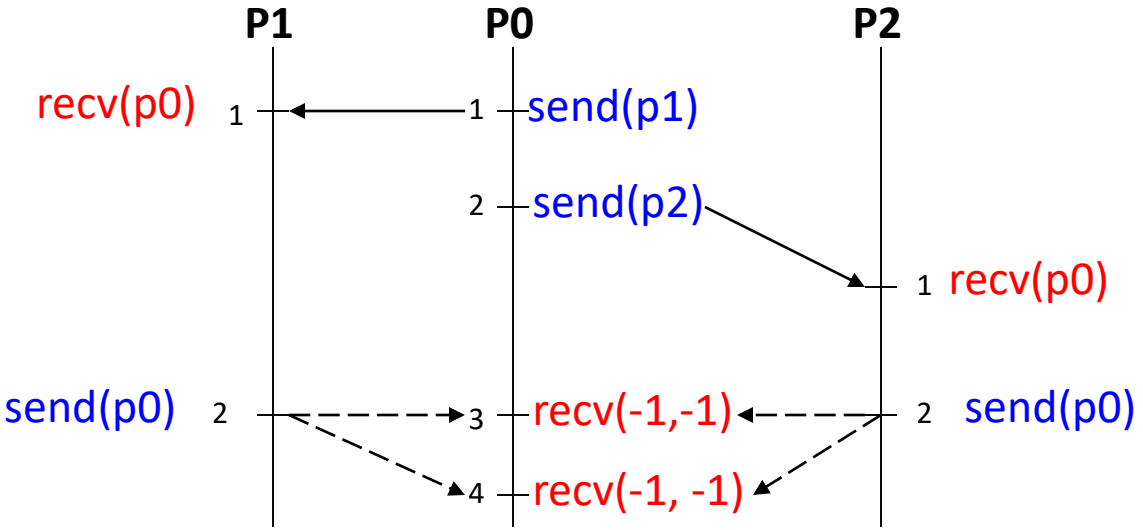
Não determinismo

Timestamp 2



Sincronizações: $(1^0,1^1)$, $(2^0,1^2)$, $(2^1,4^0)$, $(2^2,3^0)$

Não determinismo

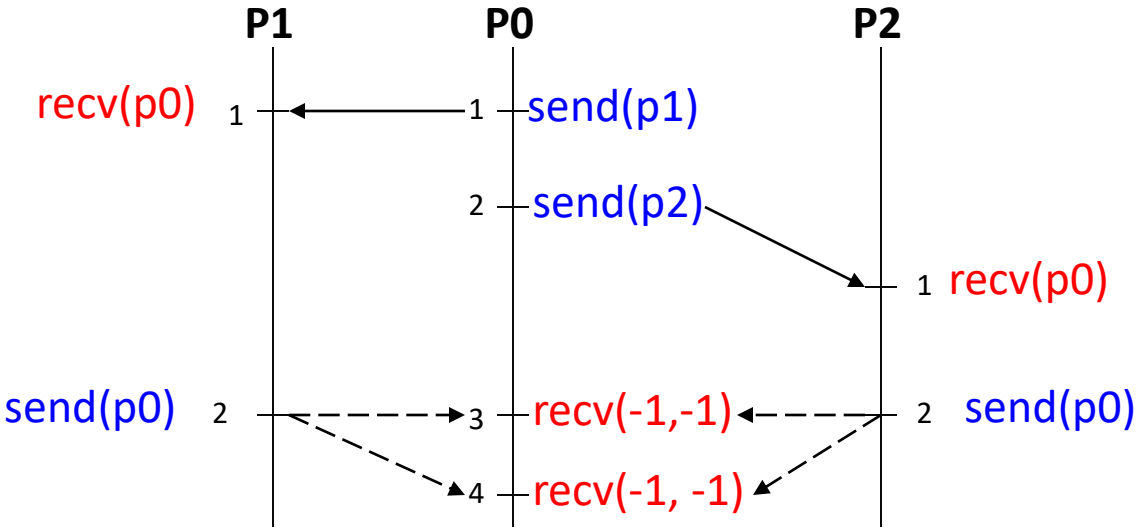


Sincronizações: $(1^0,1^1)$, $(2^0,1^2)$, $(2^1,4^0)$, $(2^2,3^0)$

Sincronizações: $(1^0,1^1)$, $(2^0,1^2)$, $(2^1,3^0)$, $(2^2,4^0)$

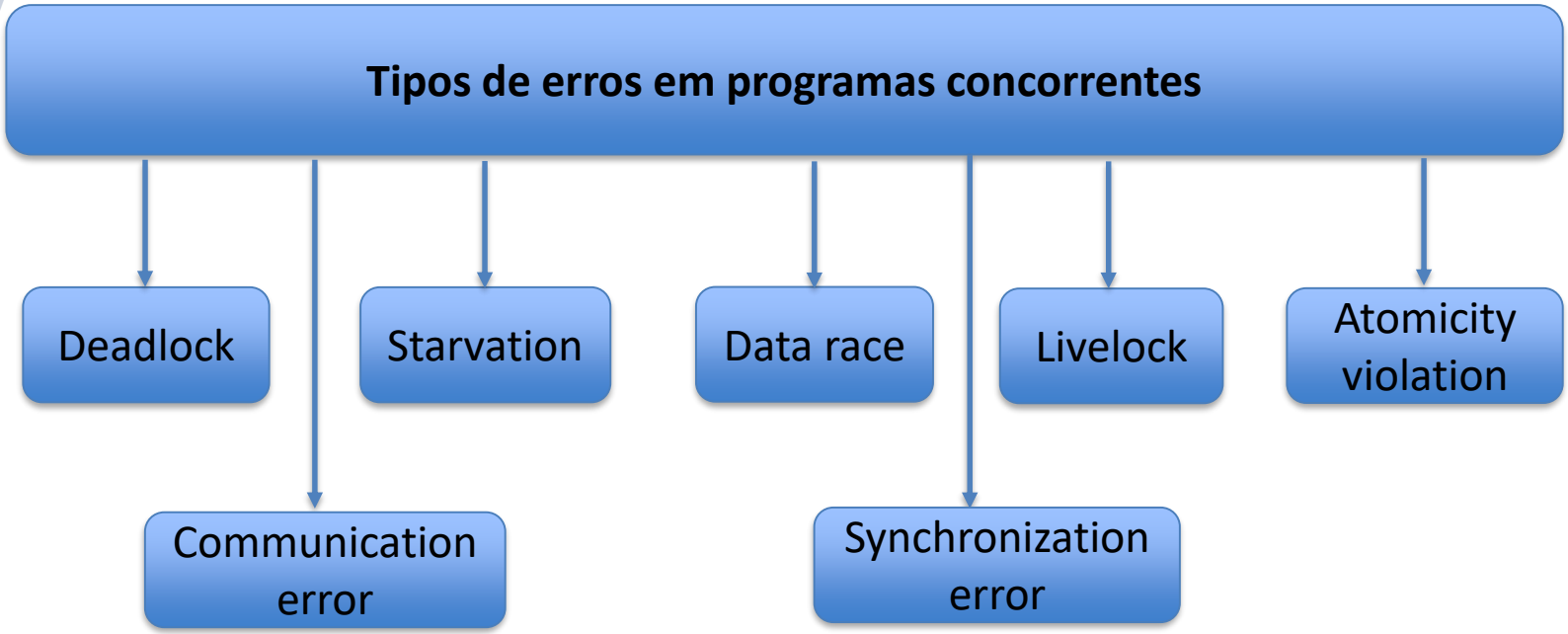
?

Não determinismo



Duas comunicações possíveis:

- Como saber qual ocorreu durante a execução?
- Qual a chance de executar uma deles (em oposição a outra)?
- Como “forçar” a execução de uma delas?



Falha de observabilidade

Falha de travamento

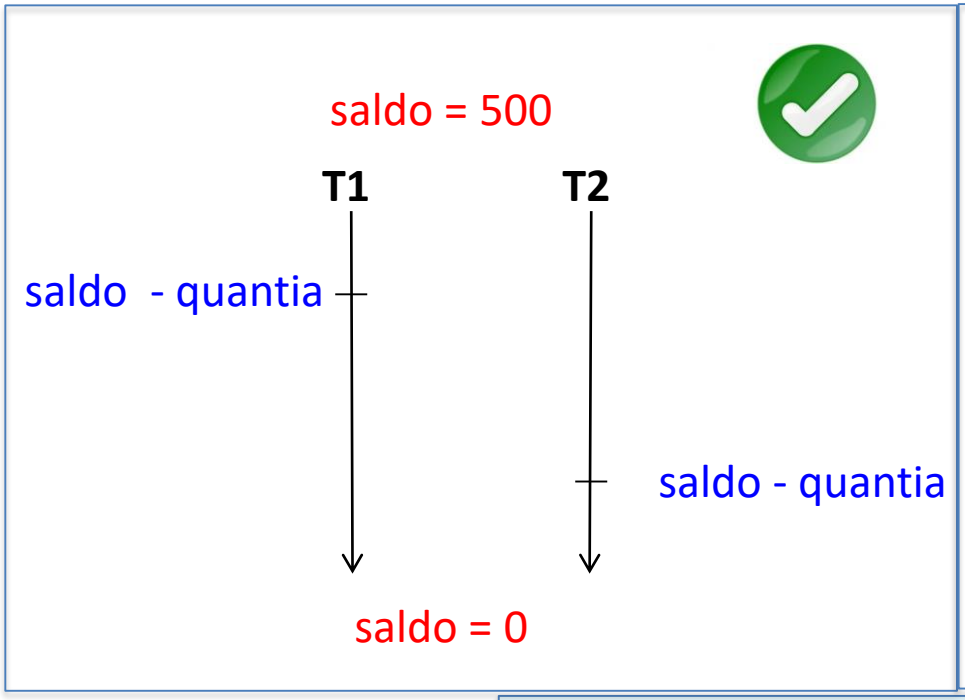
Arora, V., Bhatia, R., and Singh, M. (2016) **A systematic review of approaches for testing concurrent programs**. Concurrency Computation: Pract. Exper., 28: 1572–1611. doi: 10.1002/cpe.3711

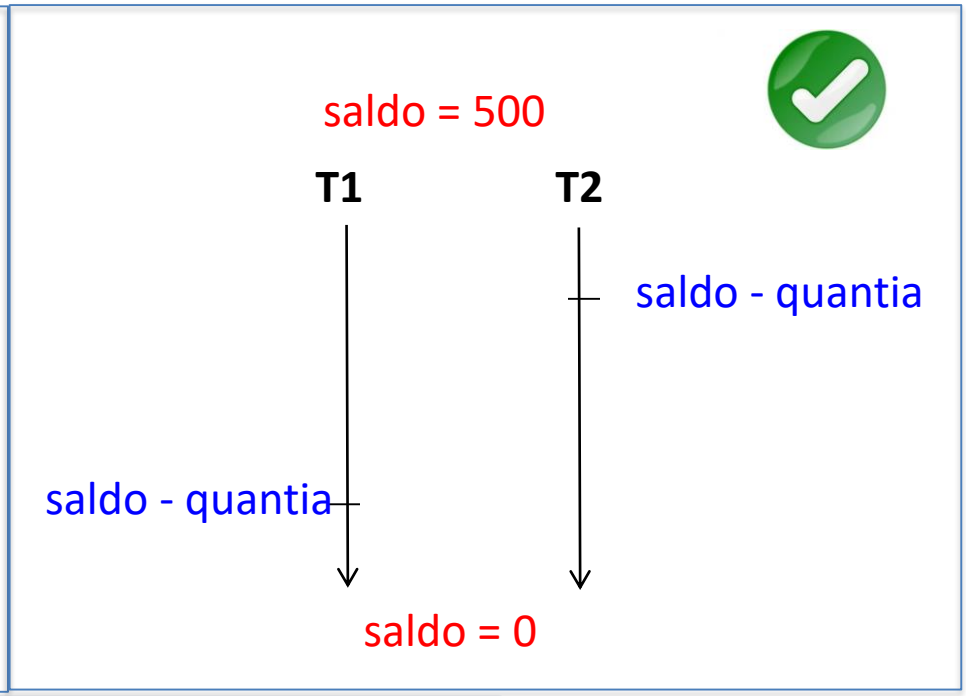
Ex: Falha de Observabilidade

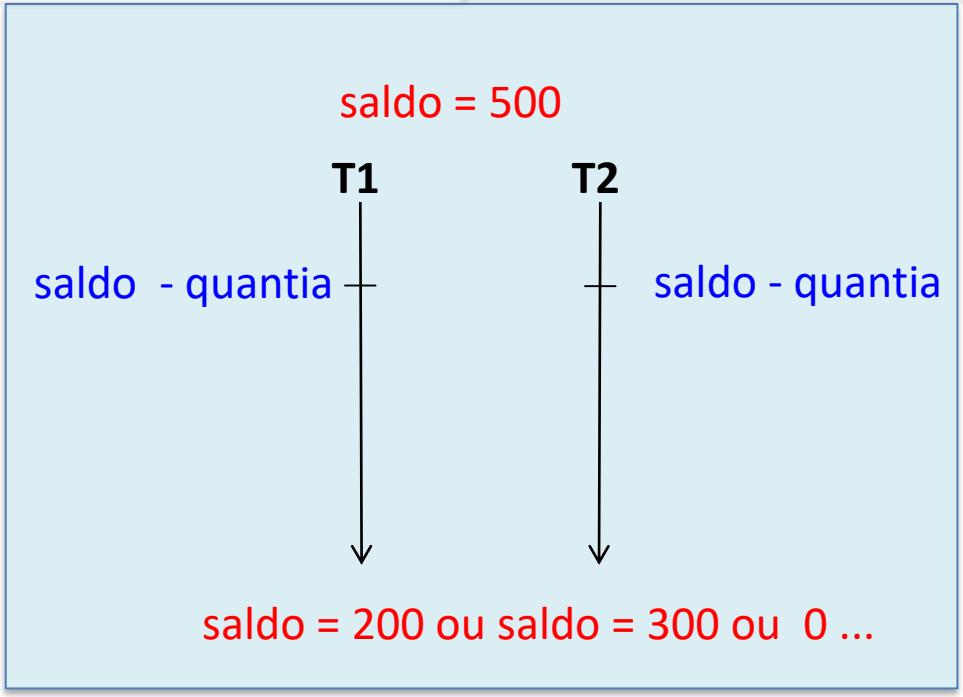
Saque() é executada por T1 e T2 e **saldo** é uma **var compartilhada**

```
bool Saque( int quantia )
 2 {
 3   if( saldo >= quantia )
 4   {
 5     saldo = saldo - quantia;
 6     return true;
 7   }
 8   else
 9   {
10     return false;
11   }
12 }
```

saldo= 500
T1 chama saque(300)
T2 chama saque(200)
Saldo final = ?


saldo = 500
T1
T2
saldo - quantia
saldo - quantia
saldo = 0


saldo = 500
T1
T2
saldo - quantia
saldo - quantia
saldo = 0

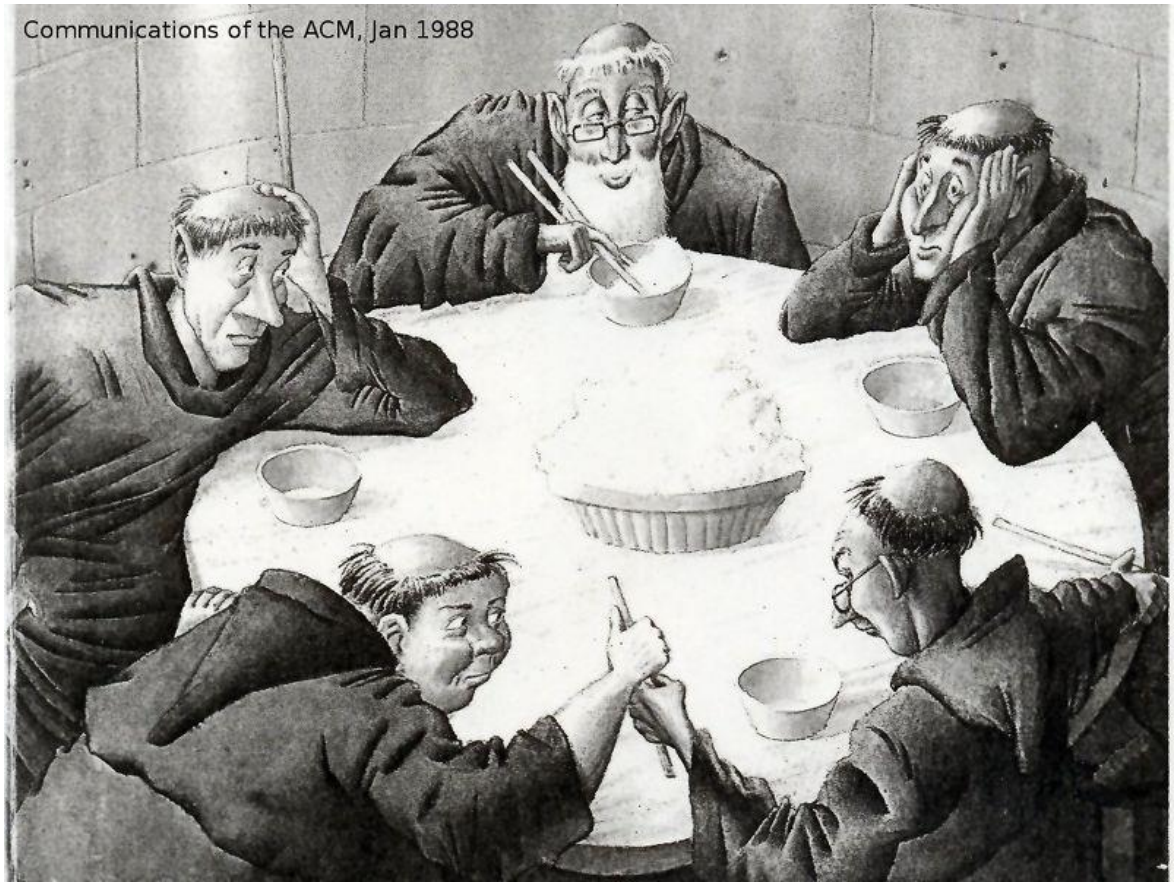

saldo = 500
T1
T2
saldo - quantia
saldo - quantia
saldo = 200 ou saldo = 300 ou 0 ...

Ex: Falha de Observabilidade

```
bool saque( int quantia )
 2 {
 3   pthread_mutex_lock(v_bloqueio);
 4   if( saldo > = quantia )
 5   {
 6     saldo = saldo - quantia;
 7     pthread_mutex_unlock(v_bloqueio);
 8     return true;
 9   }
10    else
11    {
12      pthread_mutex_unlock(v_bloqueio);
13      return false;
14    }
15 }
```

saldo= 500
T1 chama saque(300)
T2 chama saque(200)
Saldo final = 0

Falha de Travamento



http://lasdpc.icmc.usp.br/~ssc640/grad/ec2015/hashi/#

Desafios

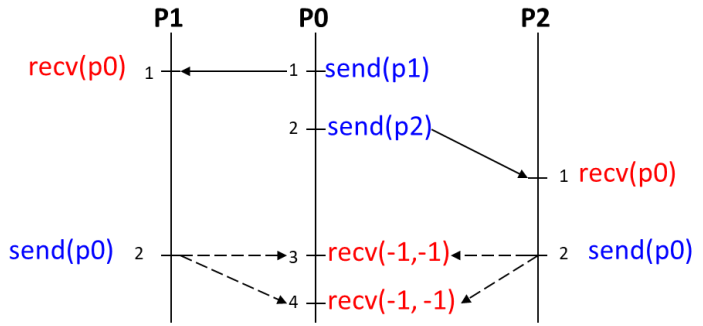
- Conceito de caso de teste:
  - *Caso de teste: (Input, Output, Sync)*

- Desafio: execução dos casos de teste
  - *Todas as sincronizações foram testadas?*
  - *Todas as saídas produzidas são corretas?*
  - *Quais sincronizações podem levar a uma falha?*

Execução de programas concorrentes

- Execução ad hoc (sem controle)

- Execução controlada
  - Determinística
  - Temporal testing



Sincronizações: $(1^0,1^1)$, $(2^0,1^2)$, $(2^1,4^0)$, $(2^2,3^0)$

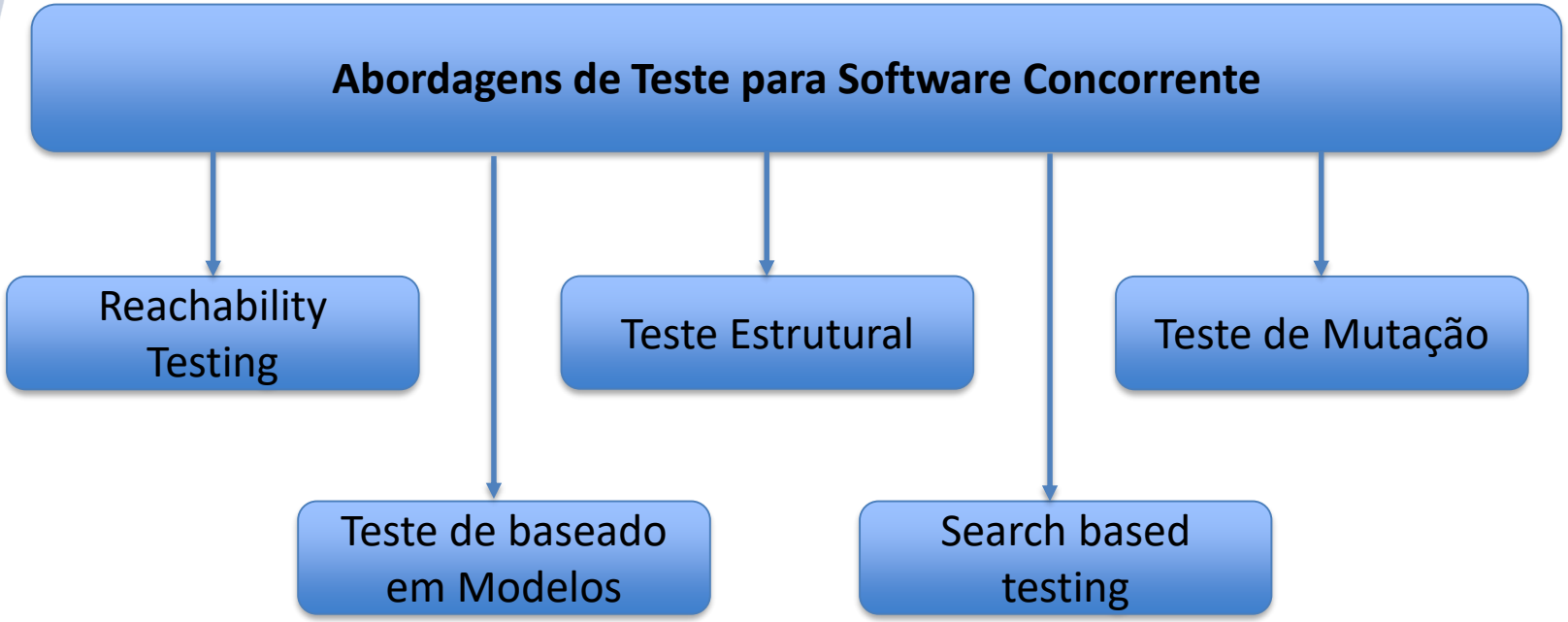
Sincronizações: $(1^0,1^1)$, $(2^0,1^2)$, $(2^1,3^0)$, $(2^2,4^0)$

?

Desafios

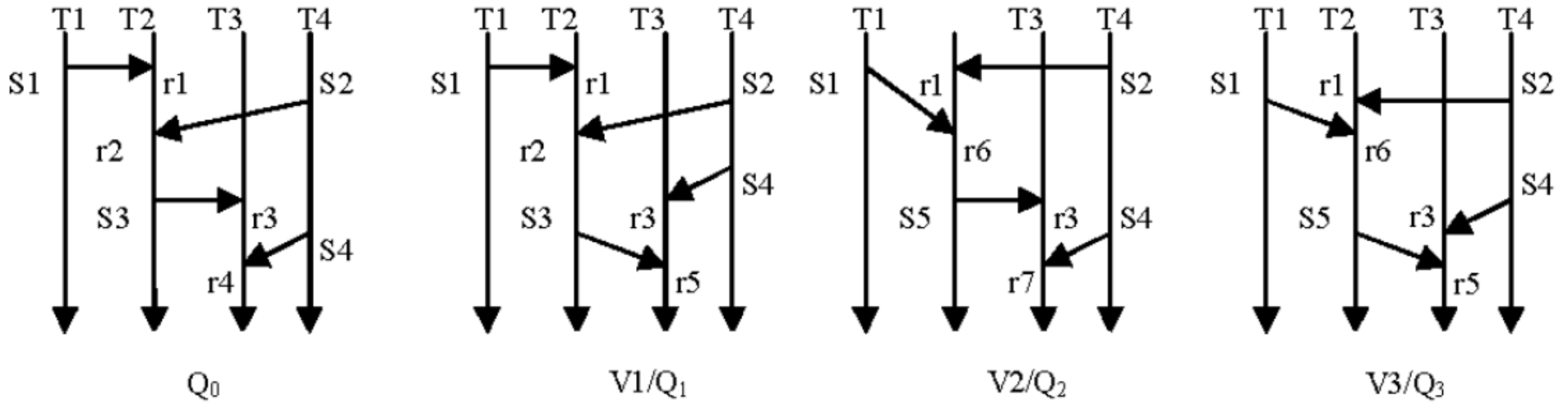
- Número elevado de requisitos de teste
  - *Interleaving dos processos ou threads*

Programa	LOC	Sync-pairs	Infeasible	All Sync
Crivo de Erastótenes	114	39	18	144
Gcd	111	14	4	126
Mmult	198	189	144	**432**
Filósofos	152	81	27	45
Qsort	480	171	78	**1404**
Jacobi	552	151	32	**1674**



Arora, V., Bhatia, R., and Singh, M. (2016) **A systematic review of approaches for testing concurrent programs**. Concurrency Computation: Pract. Exper., 28: 1572–1611. doi: 10.1002/cpe.3711

Reachability testing



G-H. Hwang, K-C. Tai and T-L. Huang. **Reachability testing: an approach to testing concurrent software**, Proc. 1st Asia-Pacific Software Engineering Conference, Tokyo, 246-255, 1994.

Y. Lei and R. H. Carver. **Reachability testing of concurrent programs**, IEEE TSE 32 (6), 382-403, 2006.

Reachability testing

- Abordagem dinâmica – não usa análise estática
- Execução parcialmente determinística
- Explosão do número de sequências de sincronização
- Vantagem: só gera sincronizações executáveis

Teste Estrutural

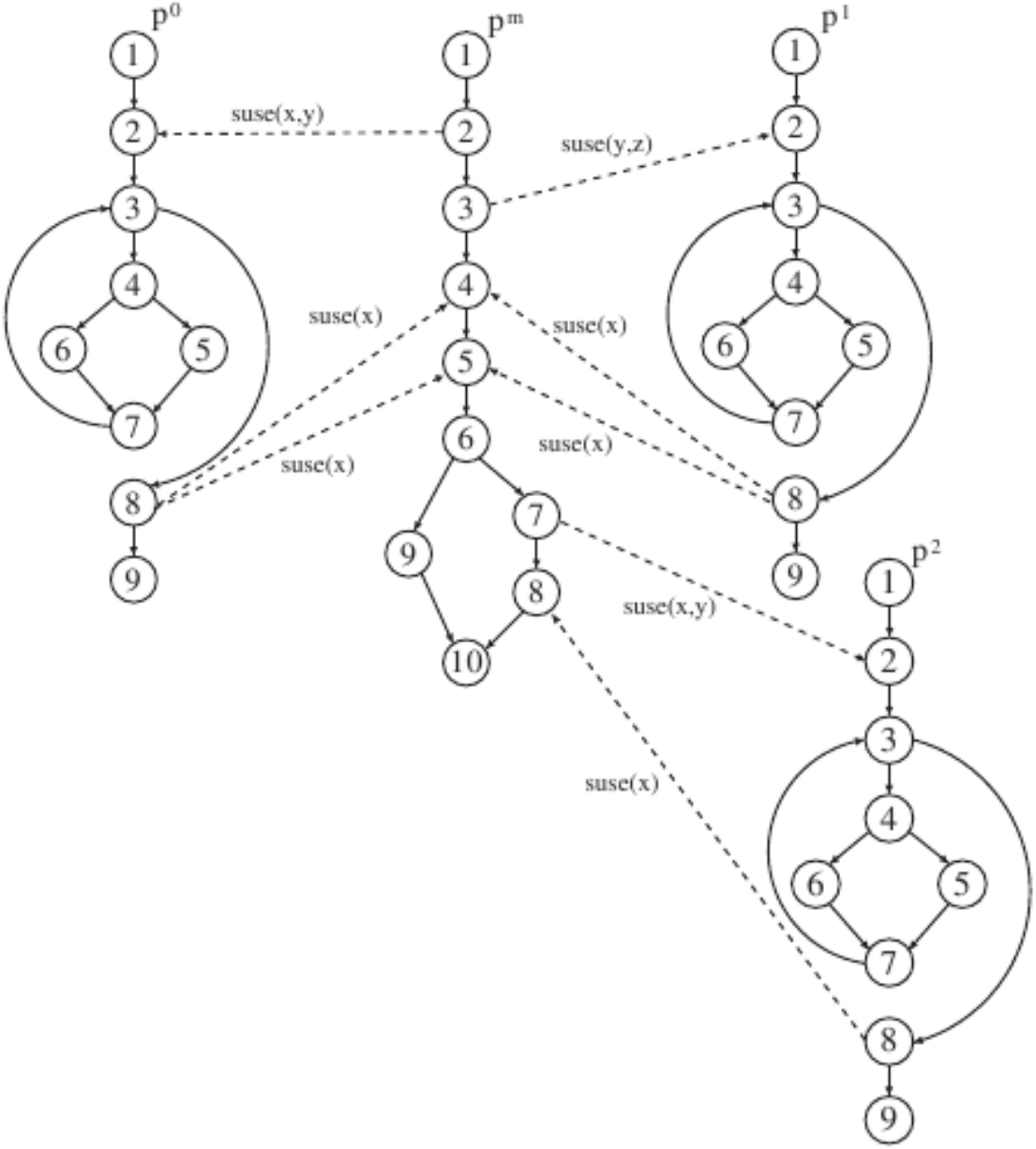
Pesquisas iniciais:

Yang, R-D. and Chung, C-G.: **Path Analysis Testing of Concurrent Programs**. Information and Software Technology. 34(1), 43–56 (1992)

Taylor, R. N. and Levine, D. L. and Kelly, C.: **Structural Testing of Concurrent Programs**. IEEE Transaction Software Engineering. 18, 3, 206–215 (1992)

Teste Estrutural

- Medida para avaliar progresso da atividade de teste
- Geração de elementos requeridos por meio de análise estática
  - Uso de um grafo para representação do programa:
    - GFCP – Grafo de Fluxo de Controle Paralelo)


p0
pm
p1
p2
suse(x,y)
suse(y,z)
suse(x)
suse(x)
suse(x)
suse(x)
suse(x,y)
suse(x)

Teste Estrutural

- Cobertura do fluxo de controle e de dados
  - Cobertura da comunicação e da sincronização entre processos/threads

- Paradigmas:
  - passagem de mensagem
  - variável compartilhada

Teste Estrutural

- Novos conceitos :
  - Paradigma de passagem de mensagens:
  - **Uso Comunicacional de variável (s-use)**: usada na mensagem entre processos:
    - send(P0, **x**); s-use(x)
    - recv(P0, **y**); s-use(y)

Teste Estrutural

- Novos conceitos:
  - Paradigma de memória compartilhada:
    - Não é possível estabelecer caminho livre de definição
    - Todo par def-uso é requerido

Ex: definição e uso (var compartilhada)

```
bool saque( int quantia )
 2 {
 3   pthread_mutex_lock(v_bloqueio);
 4   if( saldo > = quantia )
 5   {
 6     saldo = saldo - quantia;
 7     pthread_mutex_unlock(v_bloqueio);
 8     return true;
 9   }
10    else
11    {
12      pthread_mutex_unlock(v_bloqueio);
13      return false;
14    }
15 }
```

saldo= 500
T1 chama saque(300)
T2 chama saque(200)
Saldo final = 0

Opção1:

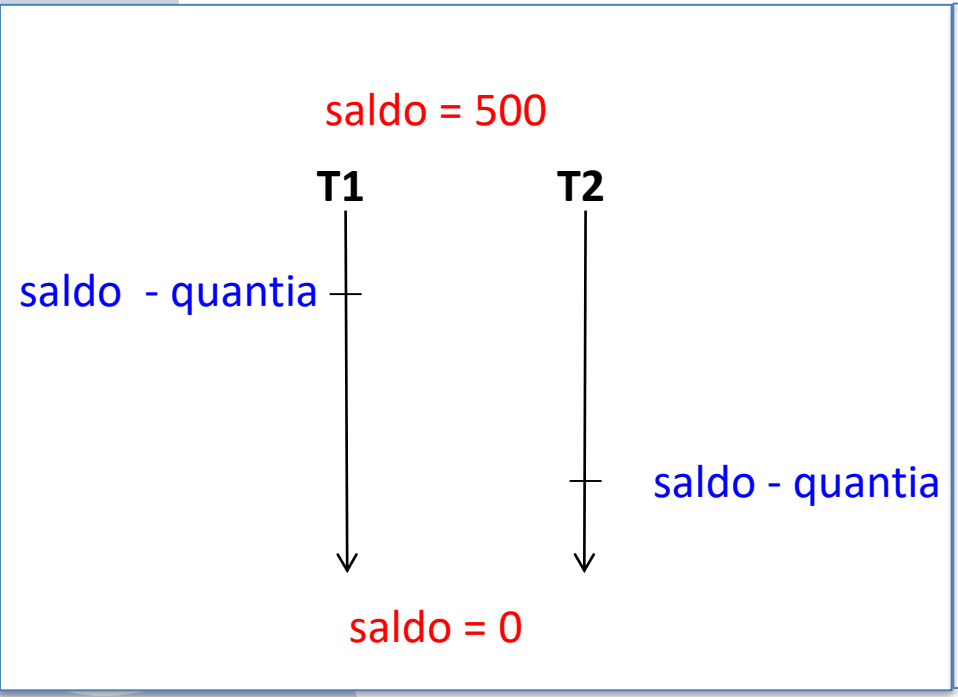
def (saldo): main()

s-uso (saldo): T1

def (saldo): T1

s-uso (saldo): T2

def (saldo): T2



Opção2:

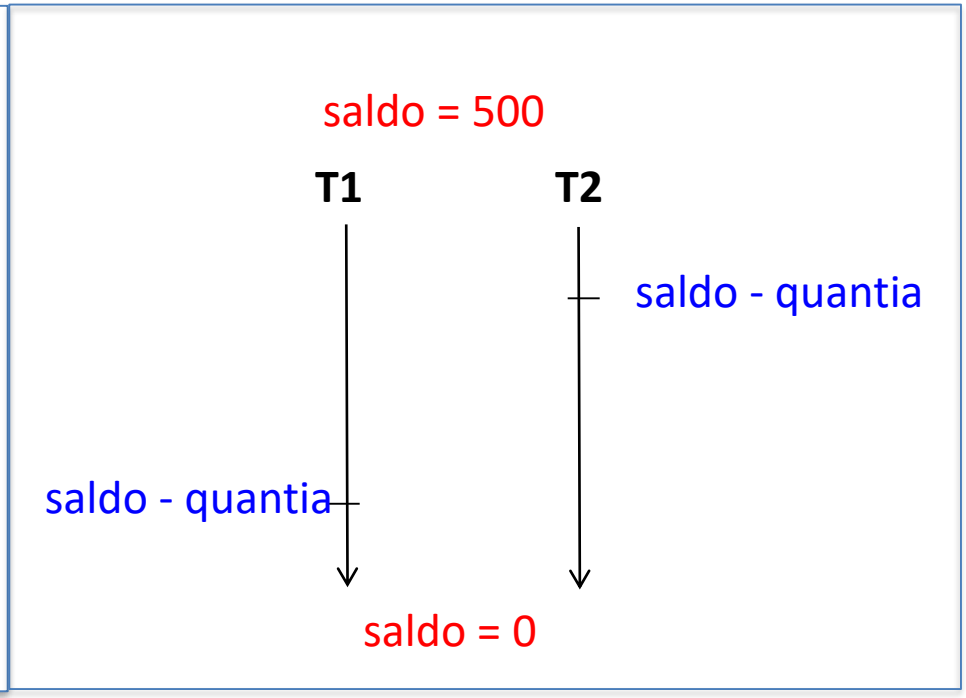
def (saldo): main()

s-uso (saldo): T2

def (saldo): T2

s-uso (saldo): T1

def (saldo): T1



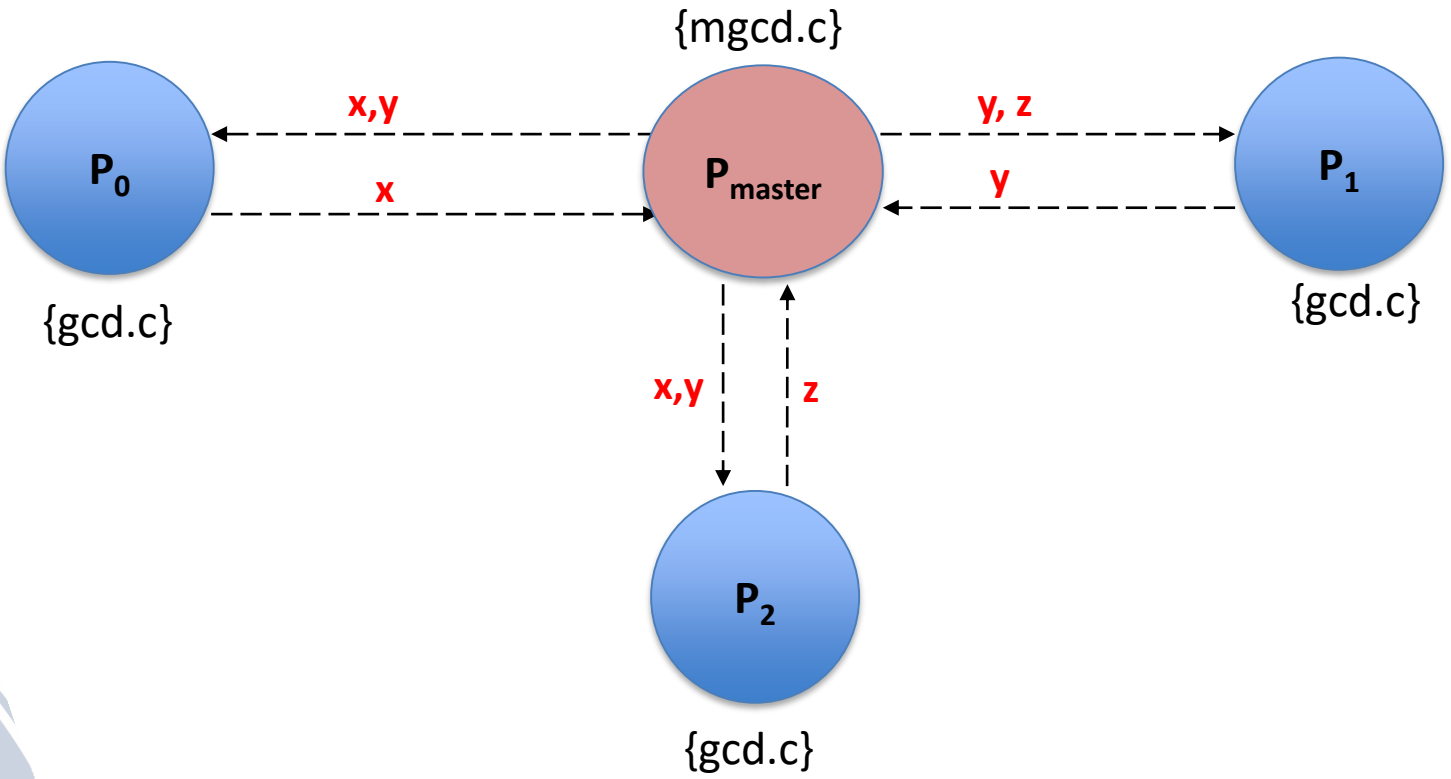
Teste Estrutural

- Critérios de fluxo de controle
  - Todos-Nós
  - Todas-Arestas
  - Todos-Nós-sync
  - Todas-Arestas-sync
- Critérios de fluxo de dados
  - Todos-usos
  - Todos-m-usos
  - Todos-s-usos

Souza, S. R. S.; Vergilio, S. R.; Souza, P. S. L.; Simao, A. S.; Hausen, A. C. **Structural testing criteria for message-passing parallel programs**. Concurrency and Computation: Practice and Experience, v. 20, p. 1893-1916, 2008.

Exemplo – Programa GCD

Programa GCD – Greatest Common Divisor
Cálculo para três números (x, y, z)



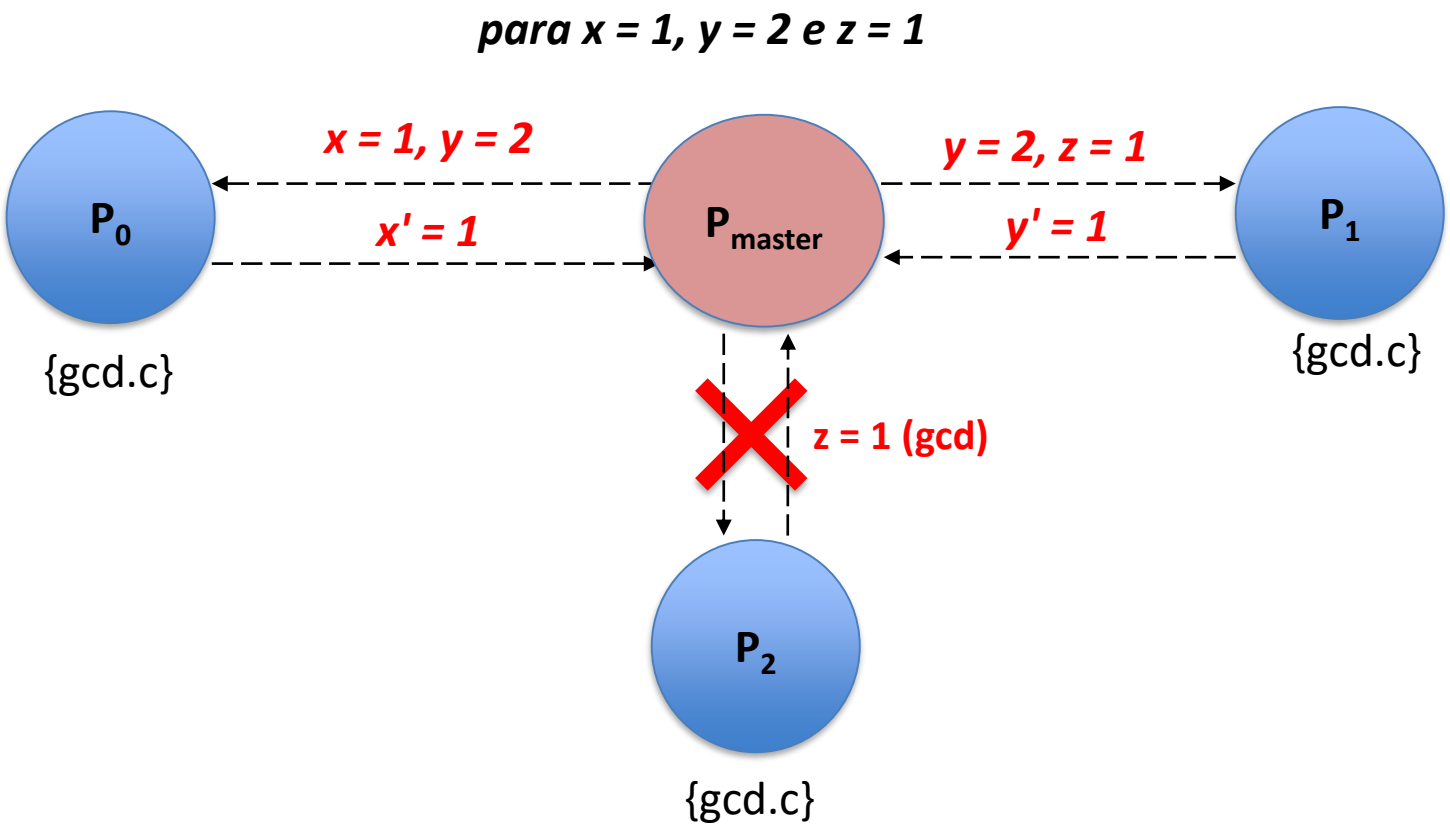
Exemplo – Programa GCD

```
/* Master program GCD - mgcd */

main() {
/*1*/  integer x,y,z, slv[3], buf[2];
/*1*/  read (x, y, z);
/*1*/  spawn_process("gcd", NULL, 3, slv);
/*2*/  buf[0] = x;   buf[1] = y;
/*2*/  send(slv[0], 2, buf);
/*3*/  buf[0] = y;   buf[1] = z;
/*3*/  send(slv[1], 2, buf);
/*4*/  recv(*, x);      // não determinismo
/*5*/  recv(*, y);      // não determinismo
/*6*/  if ( ( x>1 ) && ( y>1 ) )       {
/*7*/     buf[0] = x;    buf[1] = y;
/*7*/     send(slv[2], 2, buf);
/*8*/     recv(*, z);
/*9*/  else {
/*9*/       finalize_process(slv[2]);
/*9*/       z = 1;
/*9*/   }
/*10*/  print (z);
/*10*/  finalize();
/*10*/ }
```

```
/* Slave program GCD - gcd – 3 processos  */

main(){
/*1*/   integer pid, x, y, buf[2];
/*1*/   pid = my_father();
/*2*/   recv(pid, 2, buf);
/*2*/   x = buf[0];
/*2*/   y = buf[1];
/*3*/   while (x != y){
/*4*/         if (x<y)
/*5*/               y = y-x;
/*6*/          else
/*6*/               x = x-y;
/*7*/    }
/*8*/    send(pid,1, x);
/*9*/    finalize();
/*9*/   }
```

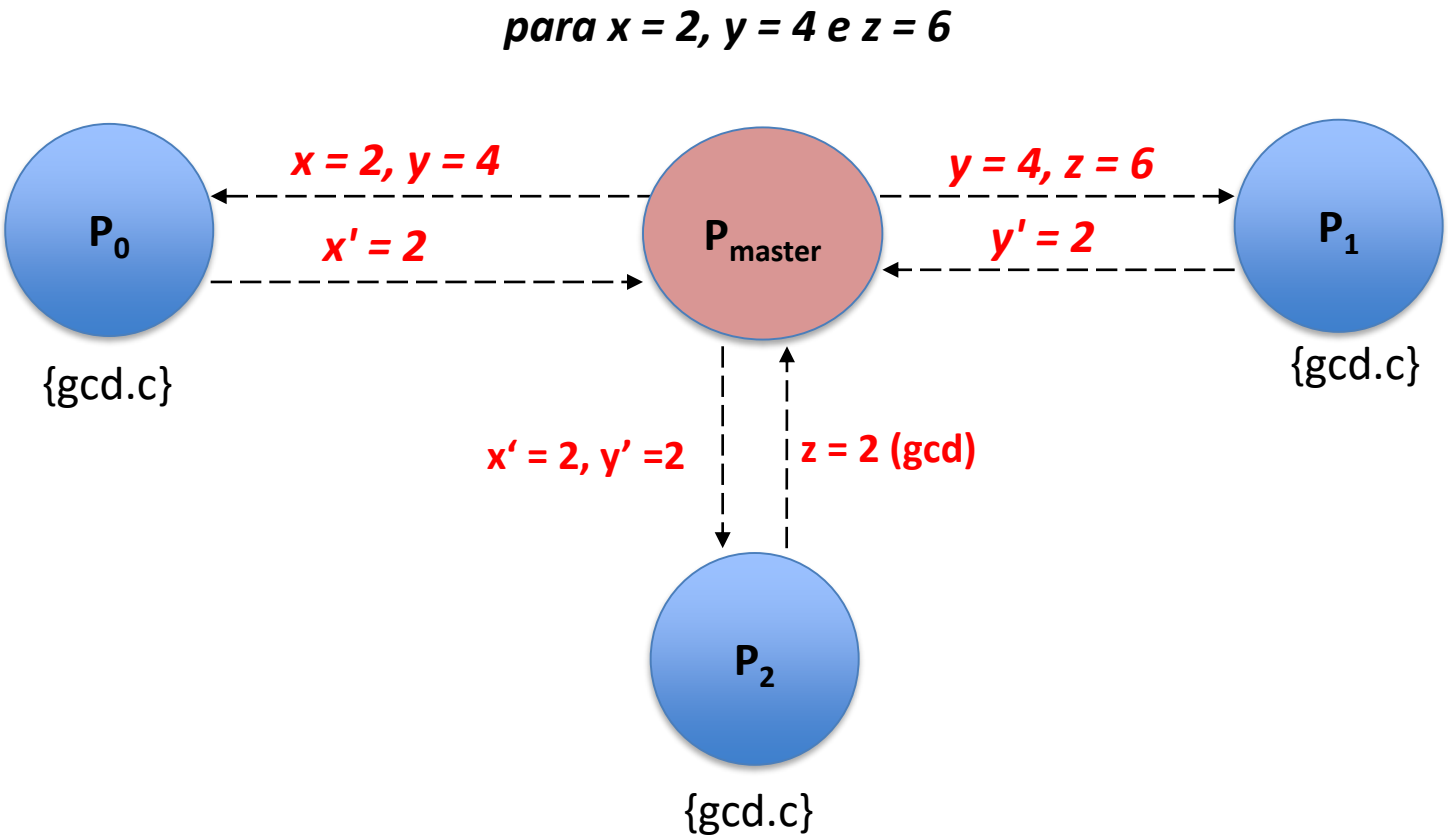
Exemplo – Programa GCD

- Programa GCD – Greatest Common Divisor



Exemplo – Programa GCD

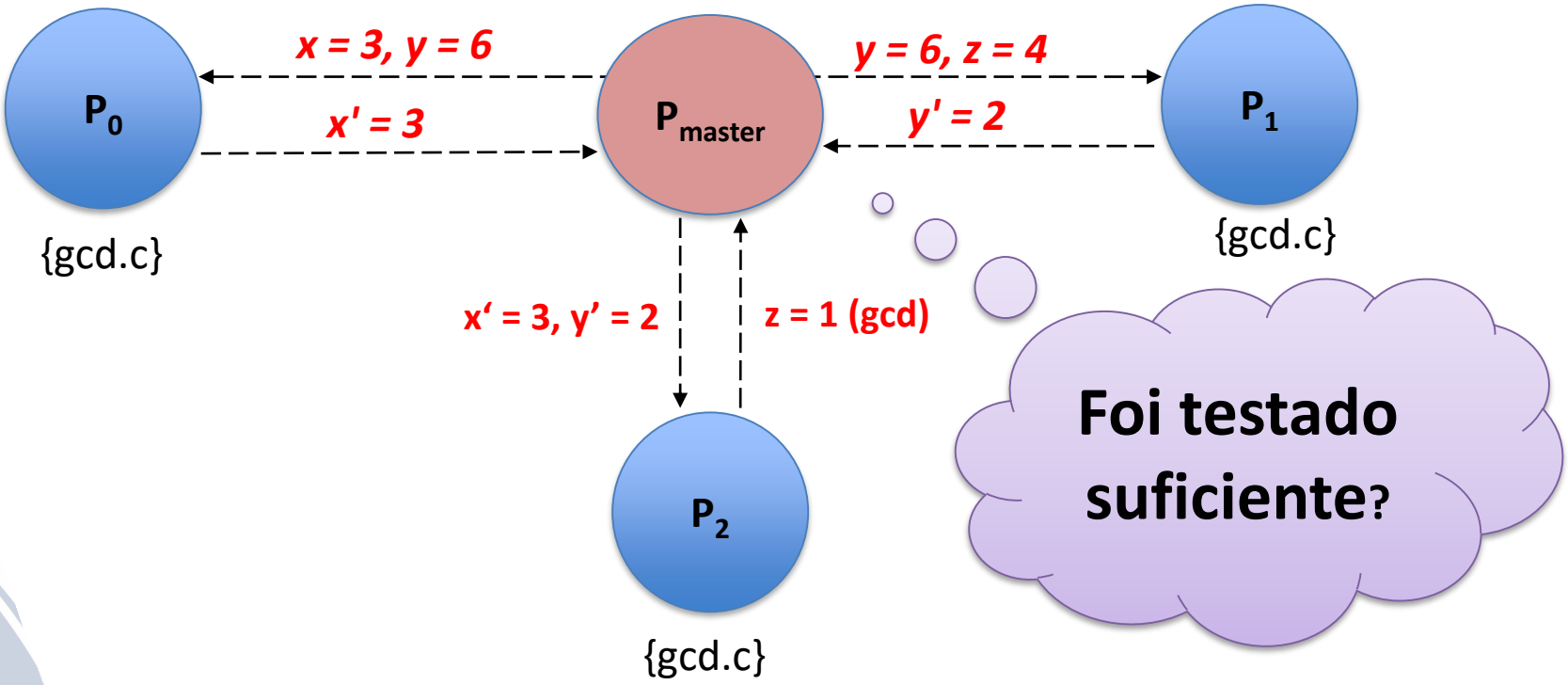
- Programa GCD – Greatest Common Divisor



Exemplo – Programa GCD

- Programa GCD – Greatest Common Divisor

para x = 3, y = 6 e z = 4



Exemplo – Programa GCD

```
/* Master program GCD - mgcd */

main() {
/*1*/  integer x,y,z, slv[3], buf[2];
/*1*/  read (x, y, z);
/*1*/  spawn_process("gcd", NULL, 3, slv);
/*2*/  buf[0] = x;   buf[1] = y;
/*2*/  send(slv[0], 2, buf);
/*3*/  buf[0] = y;   buf[1] = z;
/*3*/  send(slv[1], 2, buf);
/*4*/  recv(*, x);      // não determinismo
/*5*/  recv(*, y);      // não determinismo
/*6*/  if ( ( x>1 ) && ( y>1 ) )       {
/*7*/     buf[0] = x;    buf[1] = y;
/*7*/     send(slv[2], 2, buf);
/*8*/     recv(*, z);
/*9*/  else {
/*9*/       finalize_process(slv[2]);
/*9*/       z = 1;
/*9*/   }
/*10*/  print (z);
/*10*/  finalize();
/*10*/ }
```

```
/* Slave program GCD - gcd – 3x  */

main(){
/*1*/   integer pid, x, y, buf[2];
/*1*/   pid = my_father();
/*2*/   recv(pid, 2, buf);
/*2*/   x = buf[0];
/*2*/   y = buf[1];
/*3*/   while (x != y){
/*4*/         if (x<y)
/*5*/               y = y-x;
/*6*/          else
/*6*/               x = x-y;
/*7*/    }
/*8*/    send(pid,1, x);
/*9*/    finalize();
/*9*/   }
```

Exemplo – Programa GCD

```
/* Master program GCD - mgcd */

main() {
/*1*/  integer x,y,z, slv[3], buf[2];
/*1*/  read (x, y, z);
/*1*/  spawn_process("gcd", NULL, 3, slv);
/*2*/  buf[0] = x;   buf[1] = y;
/*2*/  send(slv[0], 2, buf);
/*3*/  buf[0] = y;   buf[1] = z;
/*3*/  send(slv[1], 2, buf);          x = 1
/*4*/  recv(*, x);
/*5*/  recv(*, y);                    y = ?
/*6*/  if ( ( x>1 ) && ( y>1 ) )      {
/*7*/     buf[0] = x;    buf[1] = y;
/*7*/     send(slv[2], 2, buf);
/*8*/     recv(*, z);
/*9*/  else {
/*9*/       finalize_process(slv[2]);
/*9*/       z = 1;
/*9*/   }
/*10*/ print (z);   z = ? (esperava 1 ☺)
/*10*/ finalize();
/*10*/ }
```

```
/* Slave program GCD - gcd */

main(){
/*1*/   integer pid, x, y, buf[2];
/*1*/   pid = my_father();
/*2*/   recv(pid, 2, buf);
/*2*/   x = buf[0];
/*2*/   y = buf[1];
/*3*/   while (x != y){
/*4*/         if (x<y)
/*5*/               y = y + x;
/*6*/          else
/*6*/               x = x - y;
/*7*/    }
/*8*/    send(pid,1, x);
/*9*/    finalize();
/*9*/   }
```

nó 5^1 como y = y + x (o correto é y = y – x)

entrada {x=1, y=2, z=1} executa o nó 5^1 e revela o defeito

Exemplo – Programa GCD

```
/* Master program GCD - mgcd */

main() {
/*1*/  integer x,y,z, slv[3], buf[2];
/*1*/  read (x, y, z);
/*1*/  spawn_process("gcd", NULL, 3, slv);
/*2*/  buf[0] = x;   buf[1] = y;
/*2*/  send(slv[0], 2, buf);
/*3*/  buf[0] = y;   buf[1] = z;
/*3*/  send(slv[1], 2, buf);
/*4*/  recv(*, x);
/*5*/  recv(*, y);
/*6*/  if ( ( x>1 ) && ( y>1 ) )      {
/*7*/     buf[0] = x;    buf[1] = y;
/*7*/     send(slv[2], 2, buf);
/*8*/     recv(*, x);
/*9*/  else {
/*9*/       finalize_process(slv[2]);
/*9*/       z = 1;
/*9*/   }
/*10*/ print (z);
/*10*/ finalize();
/*10*/ }
```

x = 1

y = 1

z = 1 (saída esperada ☹)

```
/* Slave program GCD - gcd */

main(){
/*1*/   integer pid, x, y, buf[2];
/*1*/   pid = my_father();
/*2*/   recv(pid, 2, buf);
/*2*/   x = buf[0];
/*2*/   y = buf[1];
/*3*/   while (x != y){
/*4*/         if (x<y)
/*5*/               y = y - x;
/*6*/          else
/*6*/               x = x - y;
/*7*/    }
/*8*/    send(pid,1, x);
/*9*/    finalize();
/*9*/   }
```

nó 8^m recebe msg em x : o correto seria z

entrada {x=1, y=2, z=1} não revela defeito

Exemplo – Programa GCD

```
/* Master program GCD - mgcd */

main() {
/*1*/  integer x,y,z, slv[3], buf[2];
/*1*/  read (x, y, z);
/*1*/  spawn_process("gcd", NULL, 3, slv);
/*2*/  buf[0] = x;   buf[1] = y;
/*2*/  send(slv[0], 2, buf);
/*3*/  buf[0] = y;   buf[1] = z;
/*3*/  send(slv[1], 2, buf);
/*4*/  recv(*, x);
/*5*/  recv(*, y);
/*6*/  if ( ( x>1 ) && ( y>1 ) )      {
/*7*/     buf[0] = x;    buf[1] = y;
/*7*/     send(slv[2], 2, buf);
/*8*/     recv(*, x);
/*9*/  else {
/*9*/       finalize_process(slv[2]);
/*9*/       z = 1;
/*9*/   }
/*10*/ print (z);
/*10*/ finalize();
/*10*/ }
```

x = 2

y = 2

z = 6 (esperava 2 ☺)

```
/* Slave program GCD - gcd */

main(){
/*1*/   integer pid, x, y, buf[2];
/*1*/   pid = my_father();
/*2*/   recv(pid, 2, buf);
/*2*/   x = buf[0];
/*2*/   y = buf[1];
/*3*/   while (x != y){
/*4*/         if (x<y)
/*5*/               y = y - x;
/*6*/          else
/*6*/               x = x - y;
/*7*/    }
/*8*/    send(pid,1, x);
/*9*/    finalize();
/*9*/   }
```

nó 8^m recebe msg em x: o correto seria z

entrada {x=2, y=4, z=6} executa o nó 8^m e revela defeito

Exemplo – Programa GCD

```
/* Master program GCD - mgcd */

main() {
/*1*/  integer x,y,z, slv[3], buf[2];
/*1*/  read (x, y, z);
/*1*/  spawn_process("gcd", NULL, 3, slv);
/*2*/  buf[0] = x;   buf[1] = y;
/*2*/  send(slv[0], 2, buf);
/*3*/  buf[0] = y;   buf[1] = z;
/*3*/  send(slv[1], 2, buf);
/*4*/  recv(*, x);
/*5*/  nb_recv(*, y);
/*6*/  if ( ( x>1 ) && ( y>1 ) )       {
/*7*/     buf[0] = x;    buf[1] = y;
/*7*/     send(slv[2], 2, buf);
/*8*/     recv(*, z);
/*9*/  else {
/*9*/       finalize_process(slv[2]);
/*9*/       z = 1;
/*9*/   }
/*10*/ print (z);
/*10*/ finalize();
/*10*/ }
```

x = 2

y = ?

x = 2 e y = 4

z = 2 (saída esperada ☹)

```
/* Slave program GCD - gcd */

main(){
/*1*/   integer pid, x, y, buf[2];
/*1*/   pid = my_father();
/*2*/   recv(pid, 2, buf);
/*2*/   x = buf[0];
/*2*/   y = buf[1];
/*3*/   while (x != y){
/*4*/         if (x<y)
/*5*/               y = y - x;
/*6*/          else
/*6*/               x = x - y;
/*7*/    }
/*8*/    send(pid,1, x);
/*9*/    finalize();
/*9*/   }
```

nó 5^{m} como recv não bloqueante: o correto é bloqueante

{x=2, y=4, z=6} não revela o defeito

Exemplo – Programa GCD

```
/* Master program GCD - mgcd */

main() {
/*1*/  integer x,y,z, slv[3], buf[2];
/*1*/  read (x, y, z);
/*1*/  spawn_process("gcd", NULL, 3, slv);
/*2*/  buf[0] = x;   buf[1] = y;
/*2*/  send(slv[0], 2, buf);
/*3*/  buf[0] = y;   buf[1] = z;
/*3*/  send(slv[1], 2, buf);
/*4*/  recv(*, x);
/*5*/  nb_recv(*, y);
/*6*/  if ( ( x>1 ) && ( y>1 ) )       {
/*7*/     buf[0] = x;    buf[1] = y;
/*7*/     send(slv[2], 2, buf);
/*8*/     recv(*, z);
/*9*/  else {
/*9*/        finalize_process(slv[2]);
/*9*/        z = 1;
/*9*/   }
/*10*/ print (z);
/*10*/ finalize();
/*10*/ }
```

x = 3

y = ?

x = 3 e y = 6

z = 3 (esperava 1 ☺)

```
/* Slave program GCD - gcd */

main(){
/*1*/   integer pid, x, y, buf[2];
/*1*/   pid = my_father();
/*2*/   recv(pid, 2, buf);
/*2*/   x = buf[0];
/*2*/   y = buf[1];
/*3*/   while (x != y){
/*4*/         if (x<y)
/*5*/               y = y - x;
/*6*/          else
/*6*/               x = x - y;
/*7*/    }
/*8*/    send(pid,1, x);
/*9*/    finalize();
/*9*/   }
```

nó 5m como recv não bloqueante: o correto é bloqueante

{x=3, y=6, z=4} "pode" revelar o defeito

Exemplo – Programa GCD

```
/* Master program GCD - mgcd */

main() {
/*1*/  integer x,y,z, slv[3], buf[2];
/*1*/  read (x, y, z);
/*1*/  spawn_process("gcd", NULL, 3, slv);
/*2*/  buf[0] = x;   buf[1] = y;
/*2*/  send(slv[0], 2, buf);
/*3*/  buf[0] = y;   buf[1] = z;
/*3*/  send(slv[1], 2, buf);
/*4*/  recv(*, x);
/*5*/  nb_recv(*, y);
/*6*/  if ( ( x>1 ) && ( y>1 ) )       {
/*7*/     buf[0] = x;    buf[1] = y;
/*7*/     send(slv[2], 2, buf);
/*8*/     recv(*, z);
/*9*/  else {
/*9*/        finalize_process(slv[2]);
/*9*/        z = 1;
/*9*/   }
/*10*/ print (z);
/*10*/ finalize();
/*10*/ }
```

x = 3 e y = 2

z = 1 (Saída esperada ☹)

```
/* Slave program GCD - gcd */

main(){
/*1*/   integer pid, x, y, buf[2];
/*1*/   pid = my_father();
/*2*/   recv(pid, 2, buf);
/*2*/   x = buf[0];
/*2*/   y = buf[1];
/*3*/   while (x != y){
/*4*/         if (x<y)
/*5*/               y = y - x;
/*6*/          else
/*6*/               x = x - y;
/*7*/    }
/*8*/    send(pid,1, x);
/*9*/    finalize();
/*9*/   }
```

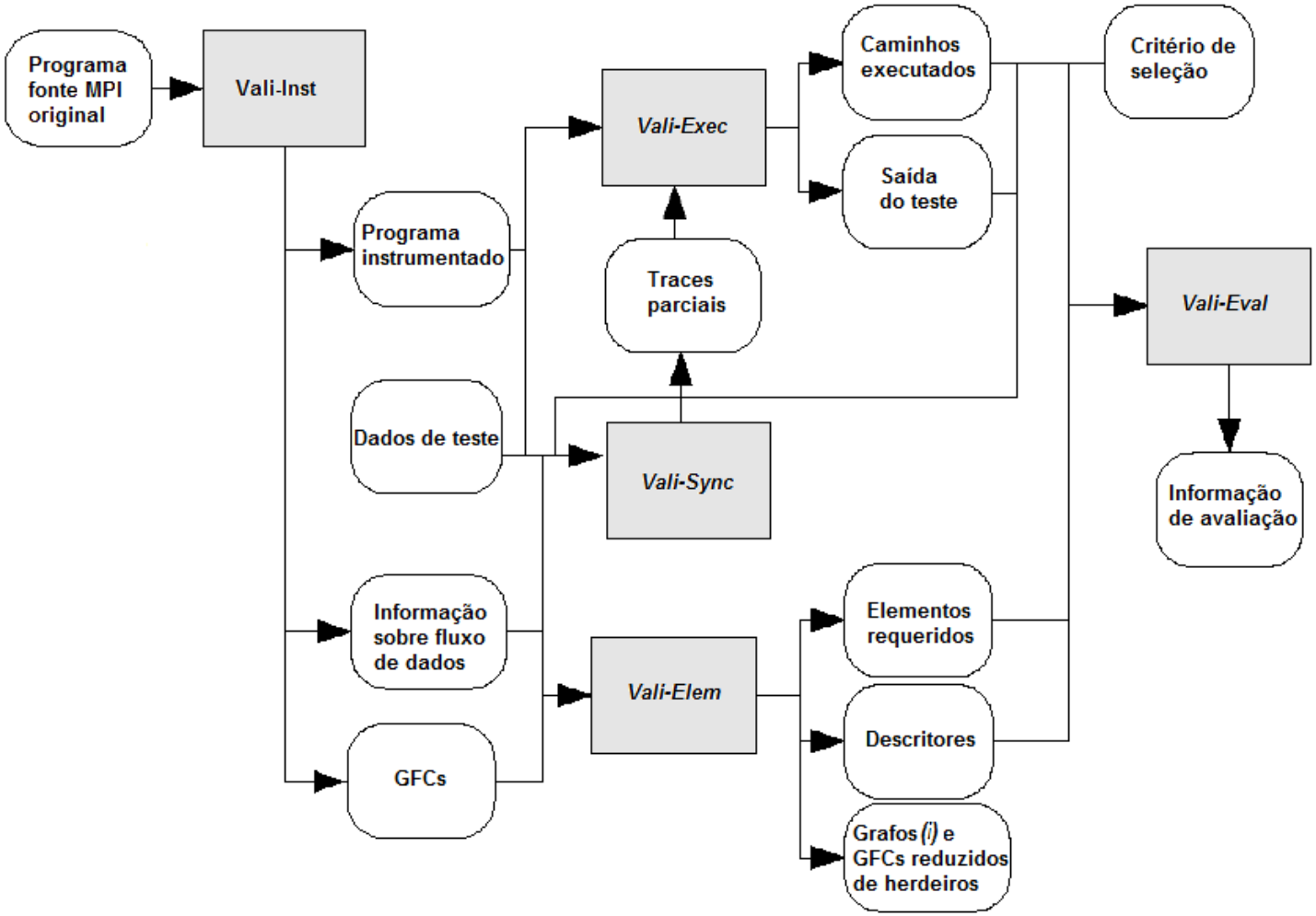
nó 5m como recv não bloqueante: o correto é bloqueante

{x=3, y=6, z=4} "pode" revelar o defeito

Ferramentas de apoio:

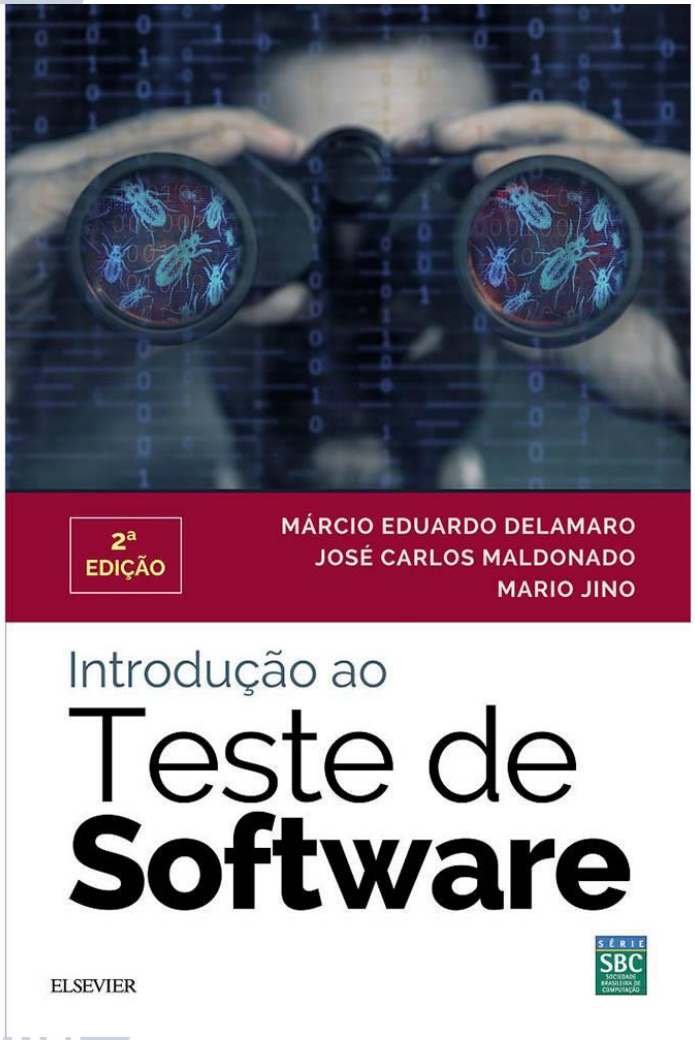
- Passagem de mensagens (C):
  - PVM (*Parallel Virtual Machine*)
  - MPI (*Message Passing Interface*)
  - **ValiMPI testing tool**
- Memória compartilhada (C):
  - Pthreads
  - **Vali-Pthread testing tool**
- Ambos paradigmas:
  - Java concorrente
  - **ValiJava testing tool**

Ferramentas ValiPar



Conclusões

- Programação Concorrente com qualidade:
  - É necessária nos dias de hoje e será também no futuro
  - Está repleta de novos desafios
    - maior demanda com novas arquiteturas
    - necessidade de maior produtividade
    - redução dos custos de manutenção
    - explorar o desempenho de pico das arquiteturas
  - Teste ainda é muito limitado – complexidade e custo



Souza, S. R. S; Souza, P. S. L.; Melo, S. M.; Silva, R. A.; Vergilio, S. R. **Teste de programas concorrentes**. In: J.C. Maldonado; M. Jino; M. E. Delamaro. (Org.). Introdução ao

Teste de Software. São Paulo, SP: Elsevier Editora Ltda, p. 231-248, 2016

Teste de Software Concorrente: quem precisa?

Simone Senger de Souza
Departamento de Sistemas de Computação
ICMC / USP
srocio@icmc.usp.br

SAST 2017 – 2º Simpósio Brasileiro de Teste de Software Sistemático e Automatizado – VIII CBSOFT – Fortaleza
Setembro/2017