OS系低レイヤな人のための Transputerとか CSP ⇒ Occam ⇒ Erlang

2011/MAY/22 (訂正版2)

たけおか

前置き
本筋とまったく関係ない

代表的な排他制御/同期のプリミティブ

- セマフォ (semaphore)
  - セマフォが得られれば、資源の使用権を得られたと考える
  - オランダ人のDijkstra(ダイクストラ)の発案
  - P/V (Wait/Signal)命令が基本
    - Waitでセマフォを得る/Signalでセマフォを返す
    - 腕木信号の用語。しかもオランダ語
  - 1bitのバイナリ・セマフォ
  - カウンティング・セマフォ (カウンタが0でなければ資源を使用可能)
  - mutexはセマフォの一種
- モニタ (monitor)
  - きわどい領域を一つの手続きにし、そこに一人(または許された数)しか入れないように、システムが制御
  - Javaが採用
- バリア(barrier), ランデブー(rendezvous)
  - 全員がそろうまで待つ
- その他
  - 交換変数(Exchange)などもあるが、あまり知られていない。プリミティブ。
- 待ちには、spin lockなどもある

より低位な排他制御/同期のプリミティブ

- test and set 命令
  - メモリの内容をアトミックに書き換え、書き換え前の状態をテストする
    - セマフォを test and set命令で実現する
      - でも、それがそのまま、プロセスの待ちに使用できるか?
  - Read modified write命令である
    - load-storeアーキテクチャのRISCシステムにそんなものはない
    - CISCには、好んで付けられた
- LL/SC (Load and Link/Store Conditional)命令
  - (近代的な)キャッシュ・メモリの機能
  - 使用法

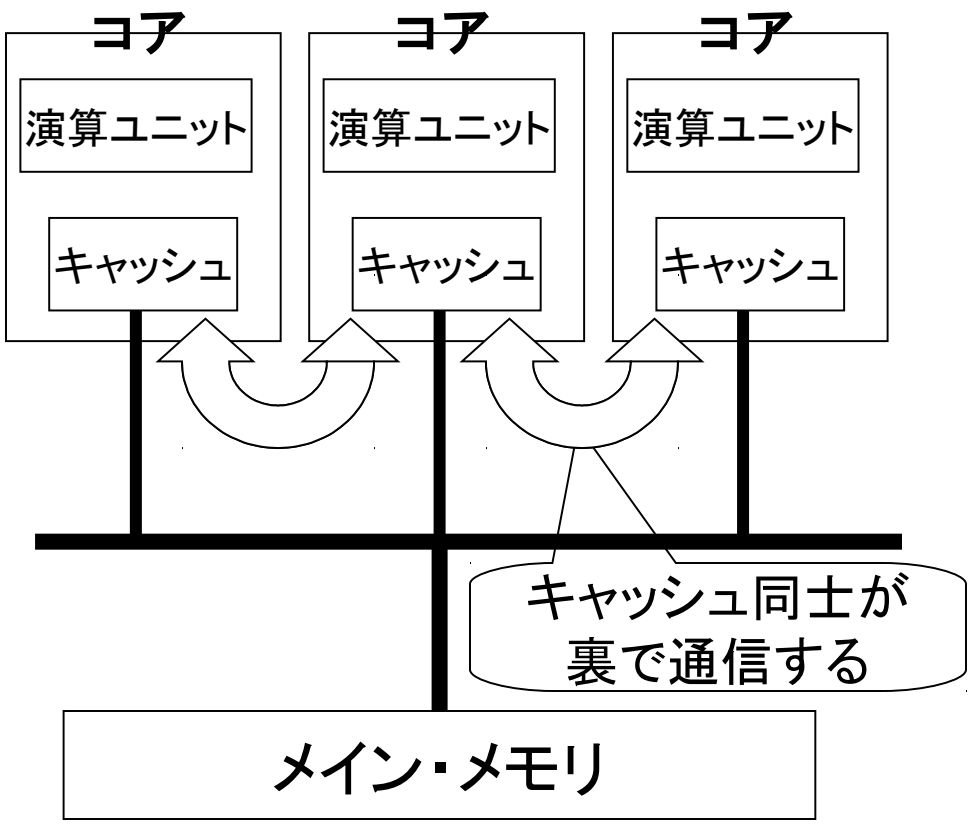
1) load linkを行って古い値を得る
2) 権利が得られそうか? 得られなければ、もっと違うレベルの待ちへ
3) 更新する値を作る
4) SCで新しい値を書き込む
5) SCの結果が失敗であれば、1へ

  - LLは、キャッシュにCPUコアからのアクセスがあったことを記憶。SCの実行前に、他のCPUコアでLLアクセスがあれば、SCが失敗する
  - キャッシュ同士が、LLアクセスがあったことを通信する

小規模マルチコア SMP,AMP

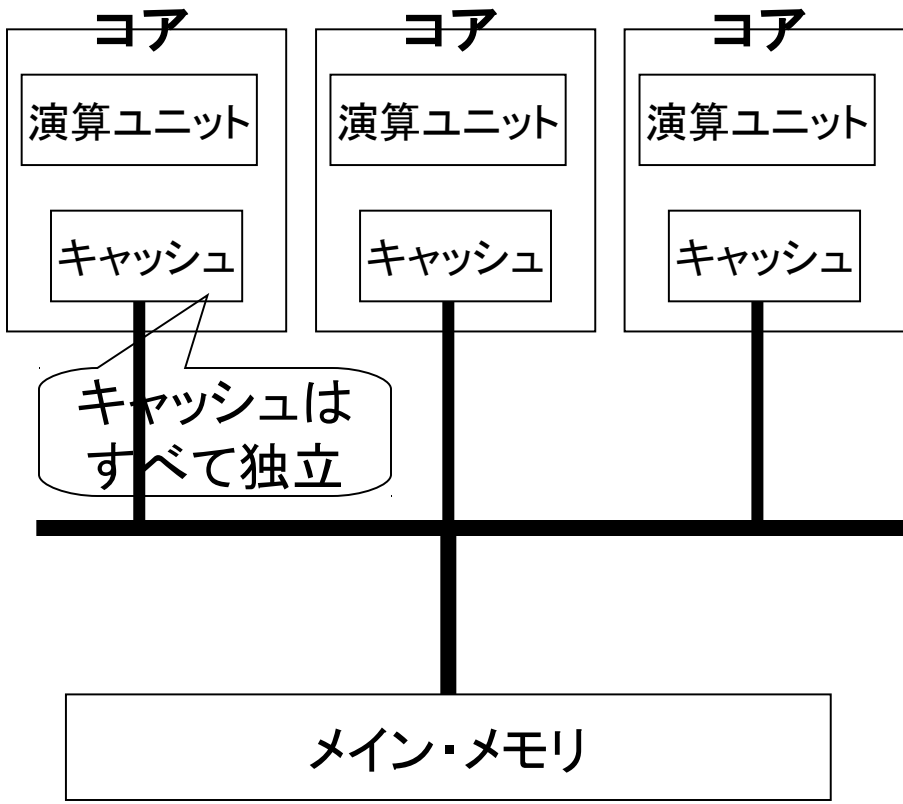
SMP

·対称型マルチ・プロセッサ
·サーバなどで昔から一般的な並列計算機
·キャッシュの一貫性(キャッシュ・コヒーレンシ)がある



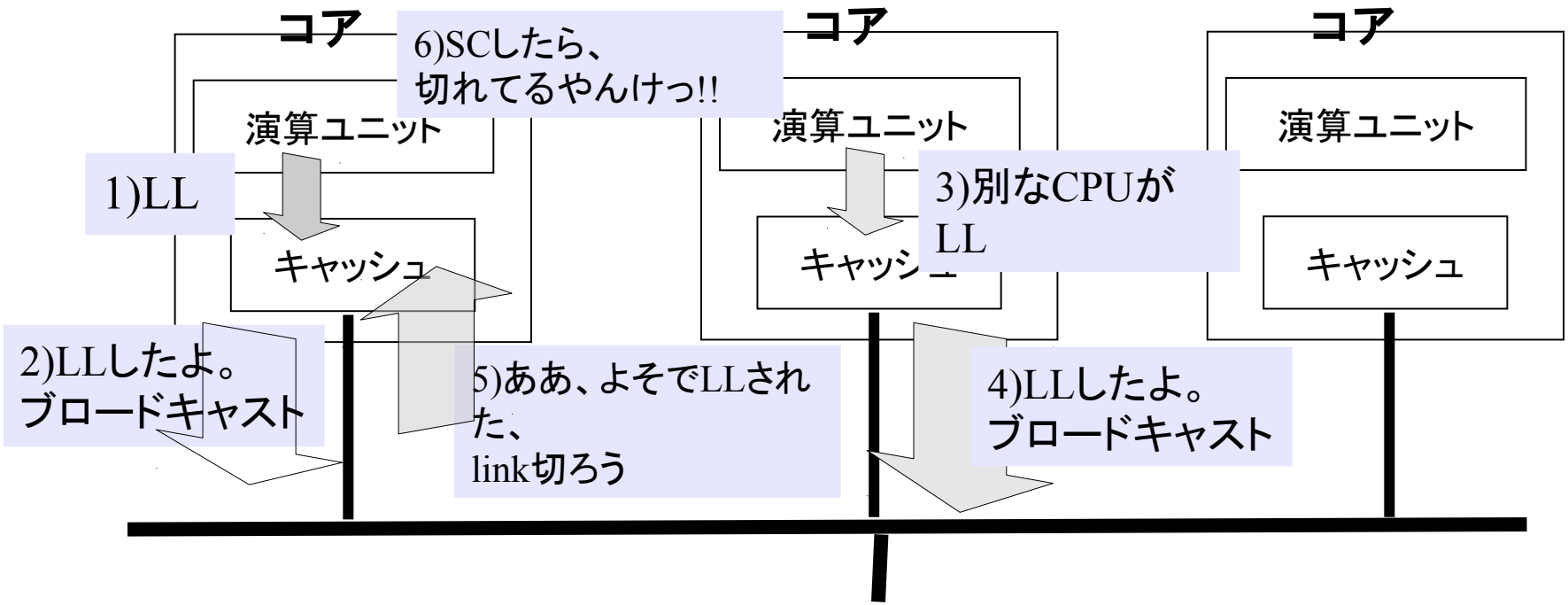
AMP

·非対称型マルチ・プロセッサ
·キャッシュの一貫性が無い
·キャッシュを意識して、コア間のデータのやりとりを行わねばならない



小規模マルチコア LL/SC

·LLは、キャッシュにCPUコアからのアクセスがあったことを記憶。
·SCの実行前に、他のCPUコアでLLアクセスがあれば、SCが失敗する
·キャッシュ同士が、LLアクセスがあったことを通信する



この辺りから
本題

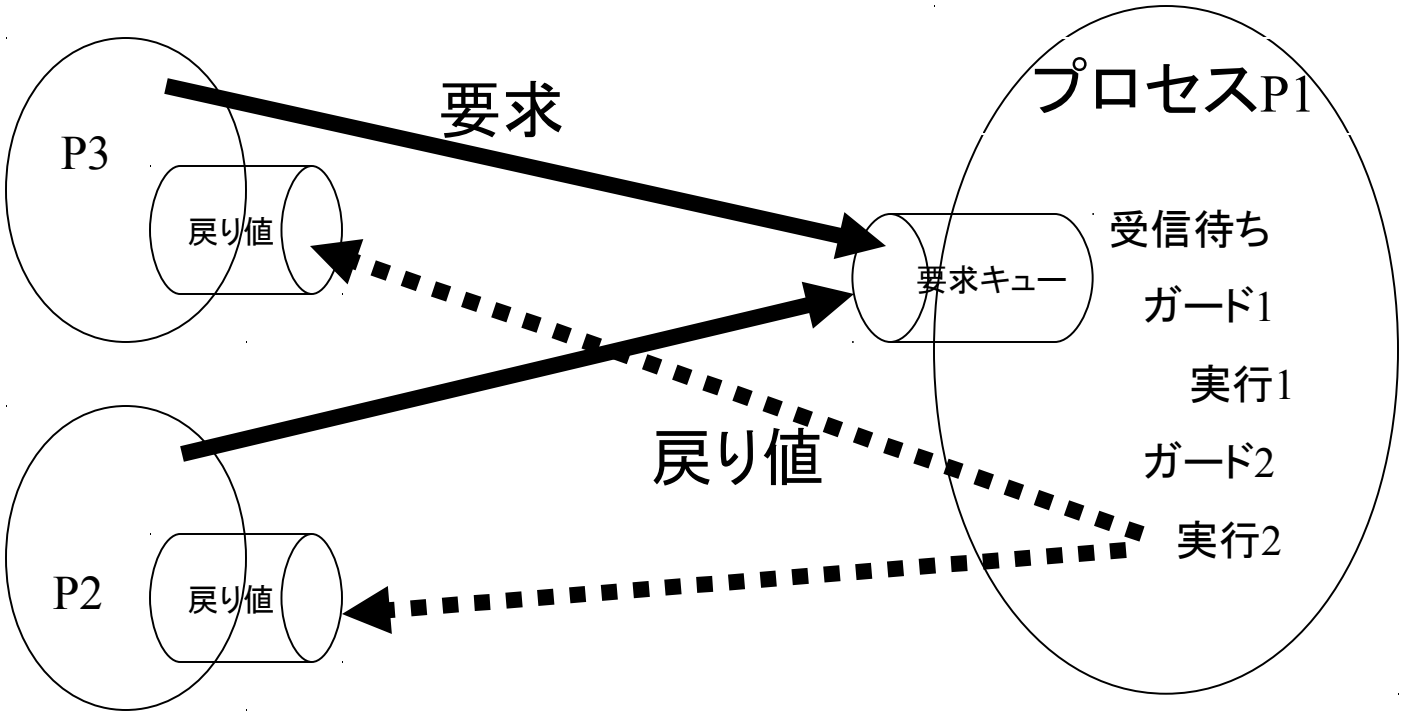
RPC

- RPC: Remote Procedure Call 遠隔手続き呼び出し
  - 遠隔にある手続きを、同期的に呼び出す
  - 遠隔の手続きは、仕事が終わると返り値をもどす
- 同期的
  - 呼ばれた側の仕事が終わるまで、呼び出し側は止まる
  - バグが出にくい
  - 素朴な実装の場合、呼ばれる側の関数は、同時に複数 入ってこないため、簡単で良い。(再入可能性の検討など不要)

- 非同期的
  - 呼ばれた側は、呼び出し側と無関係に仕事を進める
  - 終了の通知方法は?
  - 仕事の結果を置く場所は? 呼び出し側は正しくハンドリングするか

シーケンシャルなプロセスの集まりで仕事をする

サーバは
非決定的マージを行う



シーケンシャルなプログラム(プロセス)の集まりで仕事をする。
主にRPCを用いて

チャンネル通信

- CSP (Communicating Sequential processes)
  - Hoareが考えた
  - チャンネルで通信する、普通のプログラム
  - 排他制御の問題とか出ない
  - RPCに近い
    - CSPは戻り値も、チャンネル通信で返さねばならないが
  - デッドロックが起こりにくい。デッドロック発生の検出が容易
    - committed choice(コミッティド・チョイス)言語

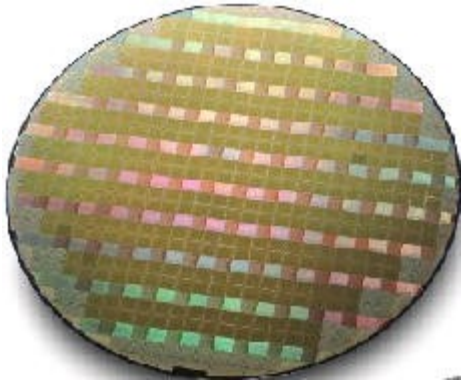
- Erlang
  - 2008年頃から、若者に流行中
  - チャンネル通信 & コミッティド・チョイス
  - パターンマッチングはProlog風、つーか、Unification
    - Unification(単一化)は、パターンマッチングと変数の双方向代入

CSP，Occam，Erlang

- CSP をもとにした現実システムがある
- Occam
  - プログラミング言語
  - 最近のTCPコネクションごとにthreadを貼り付けるのも近い考え
- Transputer
  - Occamと同時に考えられたハードウェア
  - CPUをトランジスタのごとく並べて使用。4～8本のシリアル通信ハードウェアを持つ。そのCPUを2次元のメッシュ状に配置。
  - 1990年前後に、4～16並列ぐらいの浮動小数点演算の速い機械として流行(C言語でコーディング)
- Inmos社(英)
  - TransputerとOccamを実現して販売していた
  - ST Micro社に吸収された
    - ST-10, ST-20 コアは、Transputer
  - 最近、Xmos社として復活
- Erlang
  - Erlangを発明したのは、エリクソン (ST Micro社は関係ない。発表時は誤っていた)

Transputer: 大規模並列指向CPU

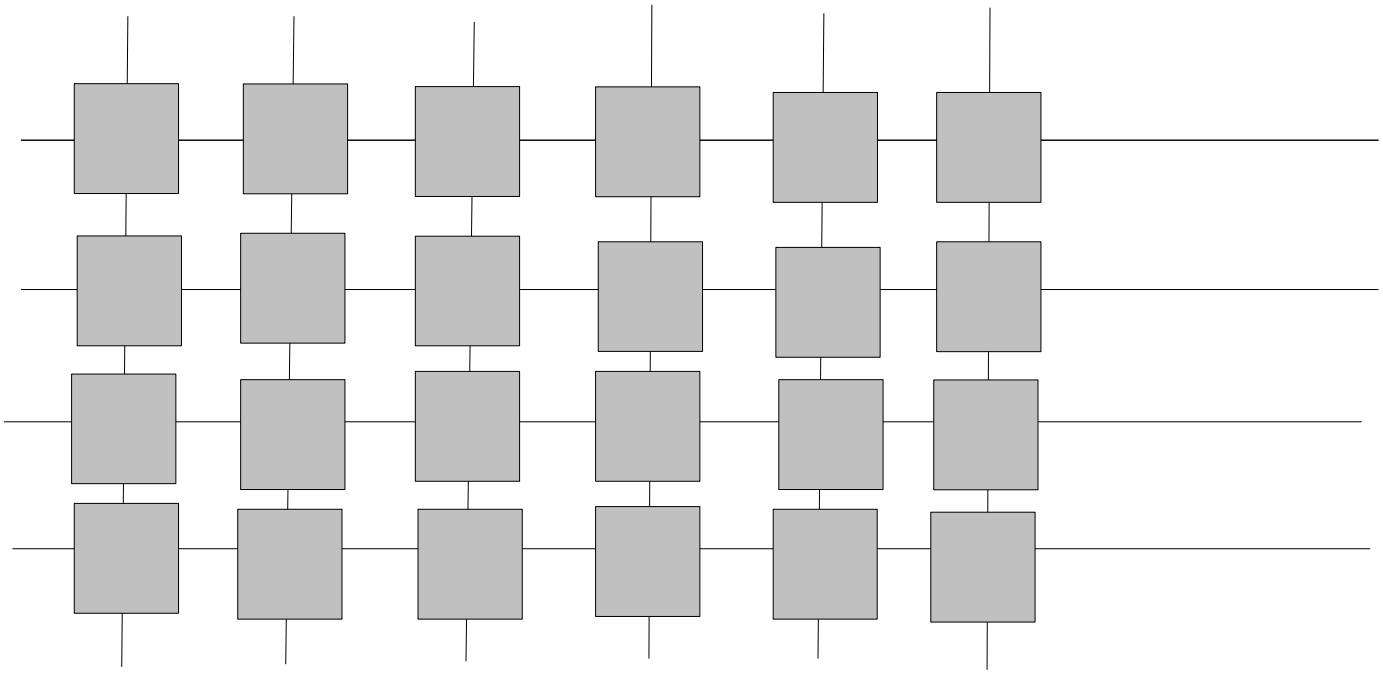
- MPP: Massively Parallel Processors
  - 大規模並列プロセッサ
  - 今、半導体企業は、組込み32bit CPU程度の素のコアならば、MPPがすぐにでも可能だと言っている
    - 1チップに、百～数百個のCPUが載る
- 1980年代末期に Transputerというものがあった
  - イギリス Inmos社
  - 百個規模のコアを1ウェハに載せ、並列計算する
  - プログラミング言語は、Occam。後にCなど
    - Tranputerは、MIMD指向
- 最近、Xmos社として復活



Transputer: 大規模並列指向CPU

·1980年代末期Transputerというものがあった

- 二次元メッシュ通信。近傍の4つのCPUとシリアル通信。
- 論理的なチャンネル通信を、そのままハードウェアに持ち込んだ

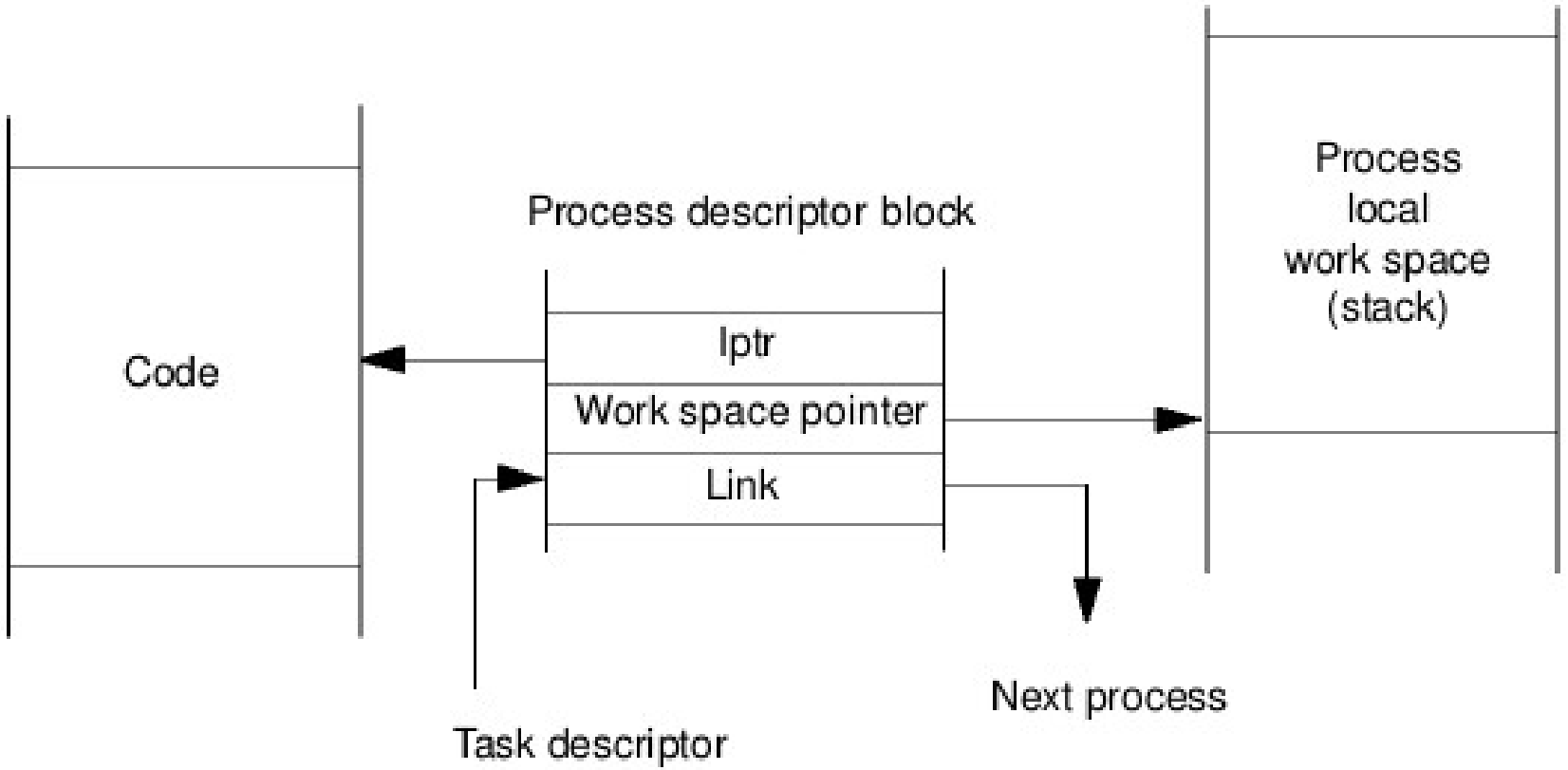


Transputer: ハードウェアでマルチタスク管理

- CPUがハードウェアで、プロセスを管理
- スケジューラをマイクロコードで実装
  - プロセス・テーブルをCPUが管理
  - レジスタのダンプ/リストアも全自動
- 通信チャンネルを待つ
  - データが来たら、プロセスがアクティベートされる
- タイムアウトでプロセス・スイッチ
  - timeslice命令がある

Transputer: ハードウェアでマルチタスク管理

- CPUがハードウェアで、プロセスを管理
- スケジューラをマイクロコードで実装



プロセス管理構造体

Transputer: ハードウェアでマルチタスク管理

- CPUがハードウェアで、プロセスを管理

スケジューラをマイクロコードで実装

  - runp: run process
  - endp: end process
  - startp: start process
  - stopp: stop process
  - insertqueue: insert queue, runキューの先頭にプロセスを入れる

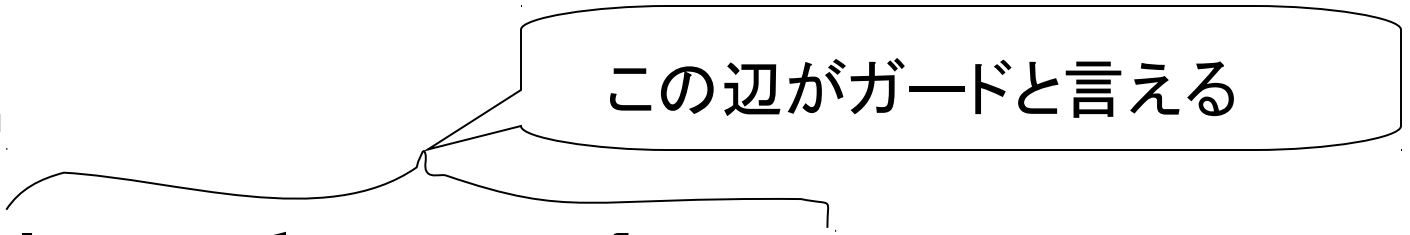
- 通信チャンネルを待つ
  - altwt : alt wait, 複数のチャンネルを待ち、どれかにデータが来たら、プロセスがアクティベートされる

- タイムアウトなどでプロセス・スイッチ
  - timeslice: timesliceを起こす
  - jump 命令でスケジューリングを起こす

Committed choice

- ガードを設けて、ガードを超えたものが実行を開始する
- CSP /Occam
  - 通信もガード条件の一種
- GHC: Guarded Horn Clause
  - 上田さんが考えた。
  - AND並列Prologの一種
  - Prologの節のコミット・バー ‘!’までをガードとする。
    - GHCではコミットバーは、「ガード記号」と呼ばれる
  - チャンネル通信をしていると考えるべし
  - 通常はバックトラックしない
- 受信と、コマンド解析&ディスパッチを同時に行う

Occamのプログラム断片例




```
ALT
    input1 ? packet
       output ! packet
    input2 ? packet
       output ! Packet
```

ALTがCommittedChoiceの指定プリミティブ

- 複数の候補のうち、ガード(上記では受信)を満たした、先着の一つだけが選ばれ、その実行部が実行される
- Occamのガードは通信しか書けない

Erlang

- エリクソンがハードウェア設計/検証のために作ったと言われている
- 変数を使用したチャンネル通信
- ガードがあり、ガードを超えた節が排他的に実行される
- パターンマッチングにユニフィケーションを使用
  - 節のヘッドでのパターンマッチングはprologのようだが、実行の意味は違う
- 型がない
  - 関数呼び出し時に、パターンマッチングする言語としてはMLが有名だが、MLは強い型の言語
- テレコム・アプリケーションを書いた実績あり

Erlangのプログラム例

```
rcv_messages() ->
        receive
            {foo, X} ->
                io:fwrite("foo~n"),
                ture;
            {bar, X} ->
                io:fwrite("bar~n"),
                ture;
        rcv_messages().
```

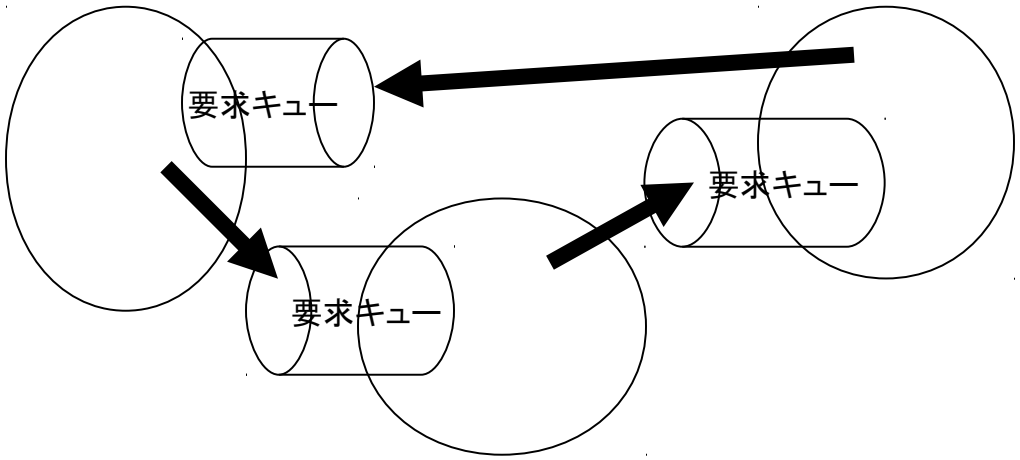
ここがガード

ガードは ‘->’の
直前まで
receiveの
受けの部分は、
単純変数,
定数、タプルなどの
パターンが書ける.
単純変数を書くと、
Occamなどの
チャンネルからの
受信変数になる

チャンネル通信(FIFO)/RPCを使おう

2010/APR/25追記

- チャンネル通信(FIFO)やRPC
  - チャンネル通信や、RPCを実行するもの(タスク)の静的な依存関係だけでデッドロックの発生がわかる
  - 複雑なシステムでも、容易にデッドロックの解析ができる
    - そもそもデッドロックが起きるように書きにくい
  - 形式的手法になじむ



※駄目(デッドロック)がすぐ判る。この例では、三すくみ(RPCじゃないし)

RPC指向の言語の排他制御

- 1つのシーケンシャル・プロセスに資源を持たせ、そのプロセスのみが資源をアクセスする
- いわゆるサーバ
- サーバに資源を持たせれば、排他制御の問題は起こらない
  - 何でもかんでも安全というわけではない
    - 例えば、サーバの中から別なサーバを呼び出すようなことをすると、デッドロックが起きるかも知れない
- 広い意味で、モニタを構成しているとも考えることができる
  - メッセージ・パッシングするオブジェクト指向風モニタ

リンク

- Transputer
  - ST20-C1 Instruction Set Reference Manual

http://www.datasheetcatalog.org/datasheet/SGSThomsonMicroelectronics/mXqxtvr.pdf

  - ST20C2/C4 Core Instruction Set Reference Manual

http://www.transputer.net/iset/pdf/st20core.pdf

- OCCAM
  - ST20C2/C4 Core Instruction Set Reference Manual

http://www.wotug.org/occam/documentation/oc21refman.pdf

マルチコア時代の並列

小規模並列

小規模マルチコア+ソフトウェアのマルチスレッド

- MIMD
  - 複数命令ストリーム 複数データ・ストリーム
  - Multiple Instruction stream Multiple Data stream
- CPUコアの数に関わらず、スレッドは作れる

- 単純な作業を、複数起動可能で無い限り
  大きな並列度を引き出すことは、難しい
- CPUコア数が大きくなると、コアを使いきれない

小規模～中規模 並列

- 計算に規則性があると楽

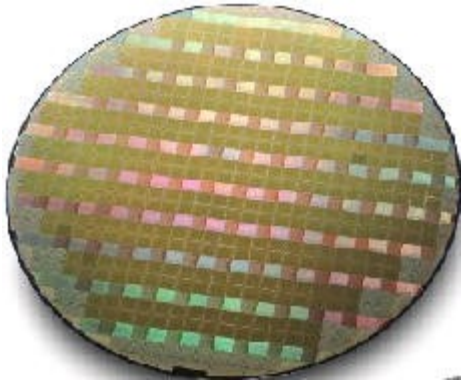
↓

- GPU計算、Intel AVX, ベクトル計算も並列計算
  - データ並列
  - SIMD計算のデータ列が長い版
    - SIMD: 単一命令ストリーム マルチ・データ・ストリーム
  - Intel AVXはベクトル計算機構=ベクトル・スパコンと同じ

- GPU計算、ベクトル計算はデータの配置が重要
  - コアを渡るデータの交換は、大規模計算のネック
  - スパコンは、コアや機械を渡るデータの交換が速い

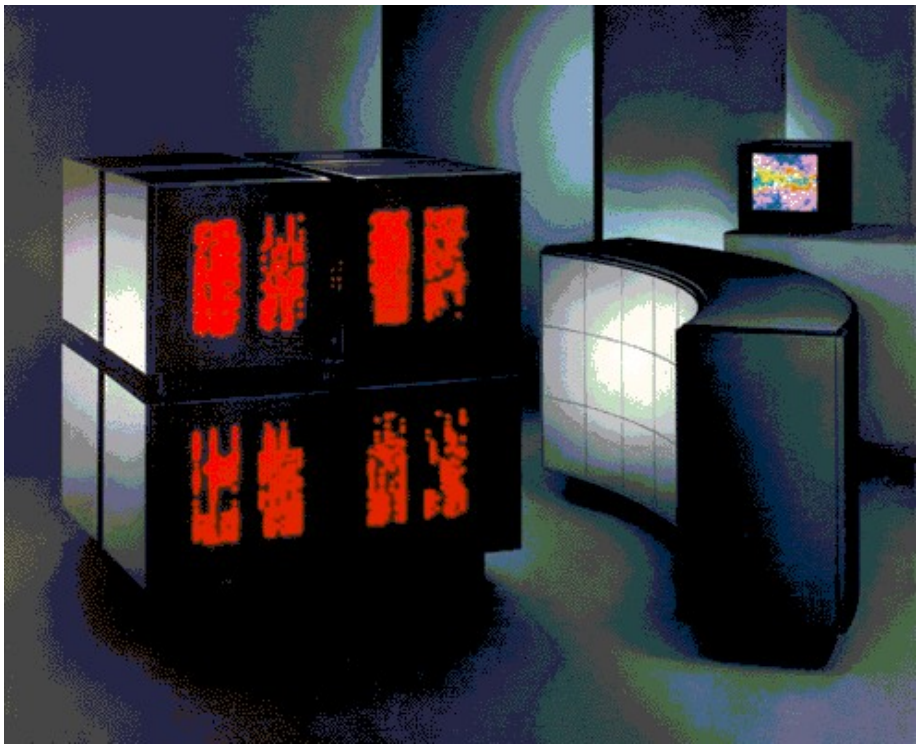
大規模 並列

- 半導体企業は、MPPがすぐにでも可能だと言っている
- MPP: Massively Parallel Processors 大規模並列プロセッサ
- 1チップに、百～数百個のCPUが載る
- 1980年代末期に Transputerというものがあった
  - イギリス Inmos社
  - 百個規模のコアを1ウェハに載せ、
    並列計算する
  - プログラミング言語は、Occam。後にCなど
  - 最近、Xmos社として復活
  - Tranputerは、MIMD指向



大規模 並列

- 大規模並列は、色んなソフトウェア(スレッド)を動かす方法では、コアを使いきれない。
- スパコンは、6万5千CPU程度が普通
  - MPI などは、一様な同じソフトウェアで、駆動
  - ベクトル計算のデータを分割して、各コアにばらまいて計算している
  - サーチエンジンなども、同じプログラムで、データが異なっている
- Googleで有名になったMap&Reduceも、1980年代末期からある
  - Connection Machine (1983年～)でメジャーに



大規模 並列

- 大規模並列は、色んなソフトウェア(スレッド)を動かす方法では、コアを使いきれない。
- 昔はSIMD
  - 命令フェッチユニット、命令デコーダなどは、システムに一つ
- スパコンは、6万コア以上が普通
  - MPI などは、同じソフトウェアで、複数のデータを同時処理
  - SPMD
    - Single Program Multiple Data stream
  - ベクトル計算のデータを分割して、各コアにばらまいて計算している

並列向き言語について

並列計算向き言語

- 関数型言語
  - 非常に流行中
  - 副作用が無い⇒他のスレッドと干渉が無い⇒並列向き
  - 竹岡は時代錯誤と考える
    - 今は、コンパイル技術で、関数型言語と同じ性質が得られる
    - コードの解析は、関数型言語で書かなくとも、行える
- Erlang
  - Occamによく似たパラダイムで動く
    - チャンネル通信
    - コミッティド・チョイス
  - チャンネル通信は、たちがいい
    - ホーアのCSP

とオシャレ(ぶった)若者に言われているが…

- 大規模な並列度が出せるか???

並列計算向き言語

- 関数型言語
- Erlang

と言われているが、大規模な並列度が出せるか???

本当は、

- SIMD, SPMDなものでないと、コアを使いきれないのでは?
  - MIMD, MPMDで、200コア回せるの???
- CSP, Erlang系の言語は、入力を待っている方が多いのでは???

並列計算向き言語

- SIMD, SPMDなものがいい(?)
- OpenMP
  - Cソースにpragmaをつける
  - なんとなく、threadが生成され、並列実行され、joinする
- MPI
  - クラスタ・スパコンの定番
  - クラスタ計算機: 同じ部屋(高速接続)で、分散計算
    - (TSUBAME は、物理的に遠いサイトも接続。異色)
  - 言語ではなく、通信/同期ライブラリ
    - 言語は、Fortran, C を使用