メニーコア対応ゲームエンジン

Many-core Game Engine Design

株式会社バンダイナムコゲームス
今給黎 隆

目次

- はじめに
- メニーコア化
- マルチプロセッサのゲームループ
- メニーコア ゲームデザイン
- バスの帯域問題

目次

- はじめに
- メニーコア化
- マルチプロセッサのゲームループ
- メニーコア ゲームデザイン
- バスの帯域問題

HWの進む方向を見て
その流れに適した
ゲームを考えよう!!

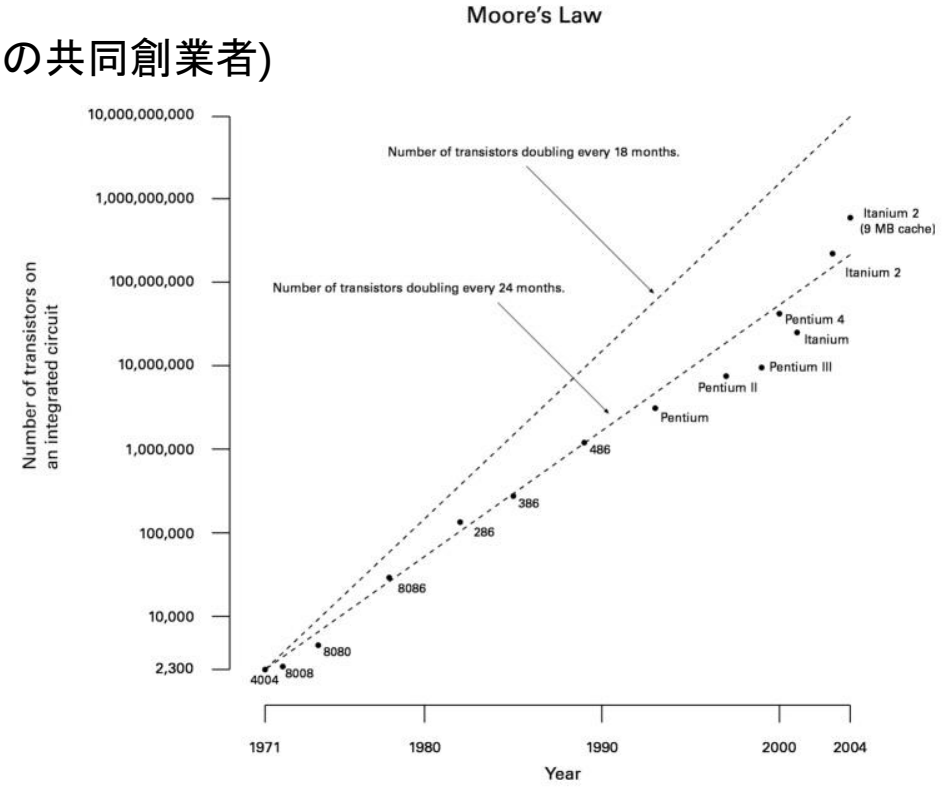
BANDAI NAMCO Games

目次

- はじめに
- メニーコア化
- マルチプロセッサのゲームループ
- メニーコア ゲームデザイン
- バスの帯域問題

ムーアの法則

- 集積回路上のトランジスタの~~速度~~数（コンピュータの処理能力のおおよその目安になる）は18か月ごとに倍になる
  - ゴードン・ムーア(インテルの共同創業者)
- 分かりやすい
- CPU業界自体を牽引
- つらくなってきた



(C) Wikipedia: Moores law.svg (Julben (2007))

ムーアの法則の限界

- 微細化
  - 原子一個のよりも微細な構造は加工できない
    - 光通信CPU?
- 電力・熱
  - ジュール熱+リーク電流

クロック数の限界

- リーク電流：微細化するほど大きくなる
  - 20%：130nmプロセス
  - 40%： 90nmプロセス
  - 50%： 65nmプロセス
- ジュール熱
  - 導線の抵抗に伴う熱

$$P \propto V^2 f \propto f^3$$

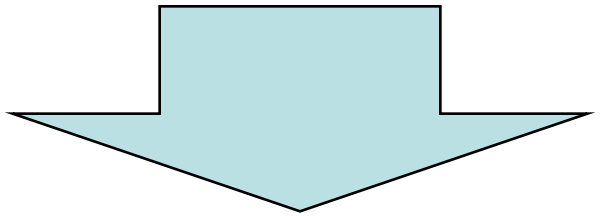
- 2001年には、2010年に
  CPUの表面温度が
  原子炉と等しくなることが
  心配されていた

周波数の向上からコア数の増加へ

- CPUの周波数の向上は頭打ち
  - 2000年代初頭以降
- CPUの性能
  - 周波数×
    サイクル当たりの命令実行数×
    命令ステップ
- 命令ステップ数の削減
  - 命令セットアーキテクチャの変更
  - コンパイラの改良
    - 「良い命令」が難しい
- 命令実行数の増加
  - SIMD化・ベクトル化
  - マルチコア化

ムーアの法則を逆に考えよう

- 集積回路上のトランジスタ数（コンピュータの処理能力のおおよその目安になる）は18か月ごとに倍になる



同じ性能のコンピュータが
二年後には半分の大きさになる

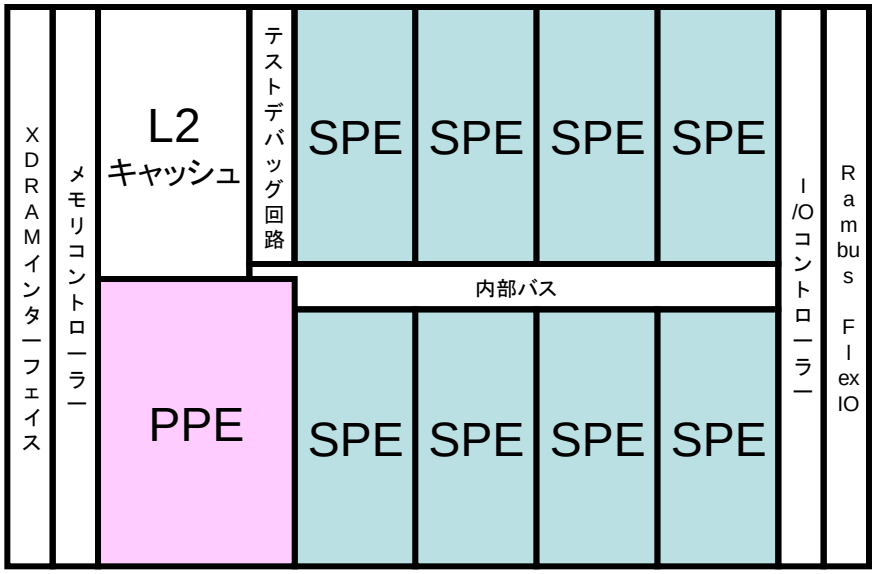
BANDAI NAMCO Games

CPU的未来予測

- 小さなコンピュータの世界(未来のコンピュータの主戦場)で、何が起きているか
  - 消費電力の削減
    - バッテリーの持続時間の延長
  - 動画は綺麗なグラフィックス性能の要求
- 両方を同時に満たすアイデア
  - CPU+GPU
    - Fusion APU(AMD)
- デスクトップにも同じ動きが
  - Clarkdale (Intel)：CPU+GPUチップ
    - 現在は別コアであるが…
- 将来のゲーム機は、CPUとGPUが融合すると考えるのが順当

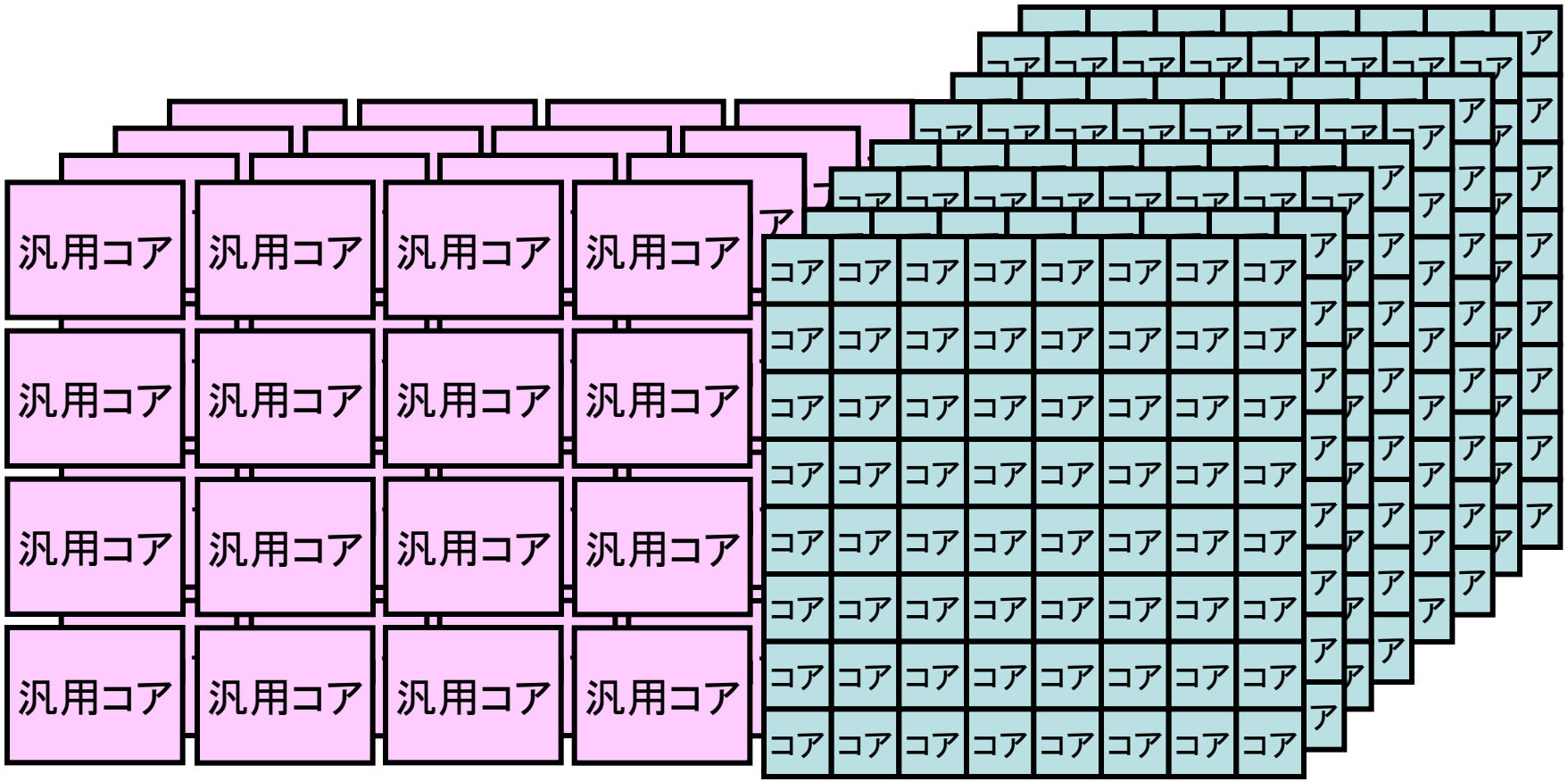
ゲーム機では

- PlayStation3
  - PPE+SPE
    - SPEは、カリングや
      コマンドバッファ生成等
      グラフィックスにも使用
- Xbox 360
  - XCGPU
    - CPUとGPUが1つのダイに



では、次世代機がどうなるか考えよう

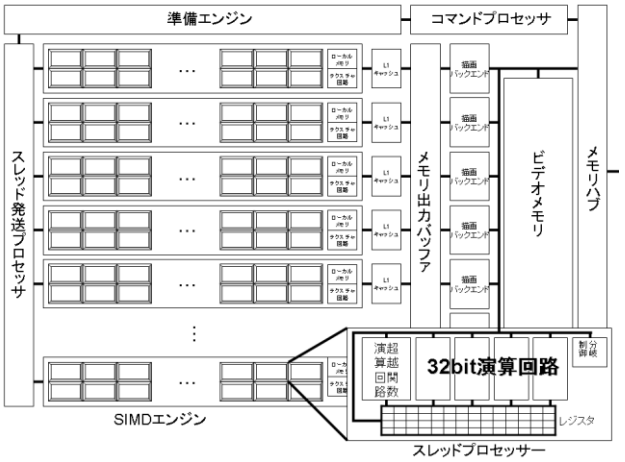
- 2年後に2倍なら10年後に64倍に!?



64汎用コア　　512特殊コア

冗談は止めておいて…

- 数百規模のコアがある時代にどのようなプログラミングを行えばよいか?
- GPUは現在も超並列プログラミング
  - 階層型の論理構造
    - Ex. ATI Radeon HD5870
    - 5命令同時実行可能なスレッドプロセッサ
      - 同じレジスタを使う
    - 20スレッド同時実行可能なSIMDエンジン
      - 同じプログラムを実行する
        - » 実際には、4SIMDエンジン(ウェイブフロント単位)で同じプログラムを実行
    - 16SIMDエンジンの同時実行
      - 1つのGPU内で1600命令が同時実行



- SIMD+データ並列化＋タスク並列化

BANDAI NAMCO Games

目次

- はじめに
- メニーコア化
- マルチプロセッサのゲームループ
- メニーコア ゲームデザイン
- バスの帯域問題

並列処理

- SIMD
  - 単一の命令で複数のデータを処理する
- フォーク - ジョイン
  - 並列性を持たせるために分割統治アルゴリズムを適用
- サブシステムごとに1つのスレッド
  - 特定のエンジンのサブシステムを別々のスレッドに割り当てて実行
- ジョブ
  - 作業を複数の小さくて比較的独立したジョブに分ける

並列処理

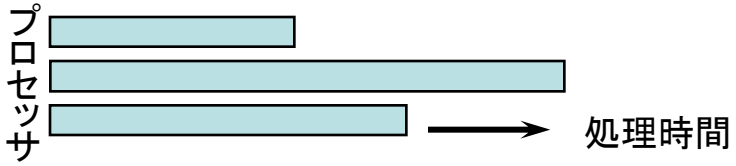
- SIMD
  - 単一の命令で複数のデータを処理する
  - 将来は並列度が上がる？
- フォーク - ジョイン
  - 並列性を持たせるために分割統治アルゴリズムを適用
- サブシステムごとに1つのスレッド
  - 特定のエンジンのサブシステムを別々のスレッドに割り当てて実行
- ジョブ
  - 作業を複数の小さくて比較的独立したジョブに分ける

マルチプロセッサのゲームループ

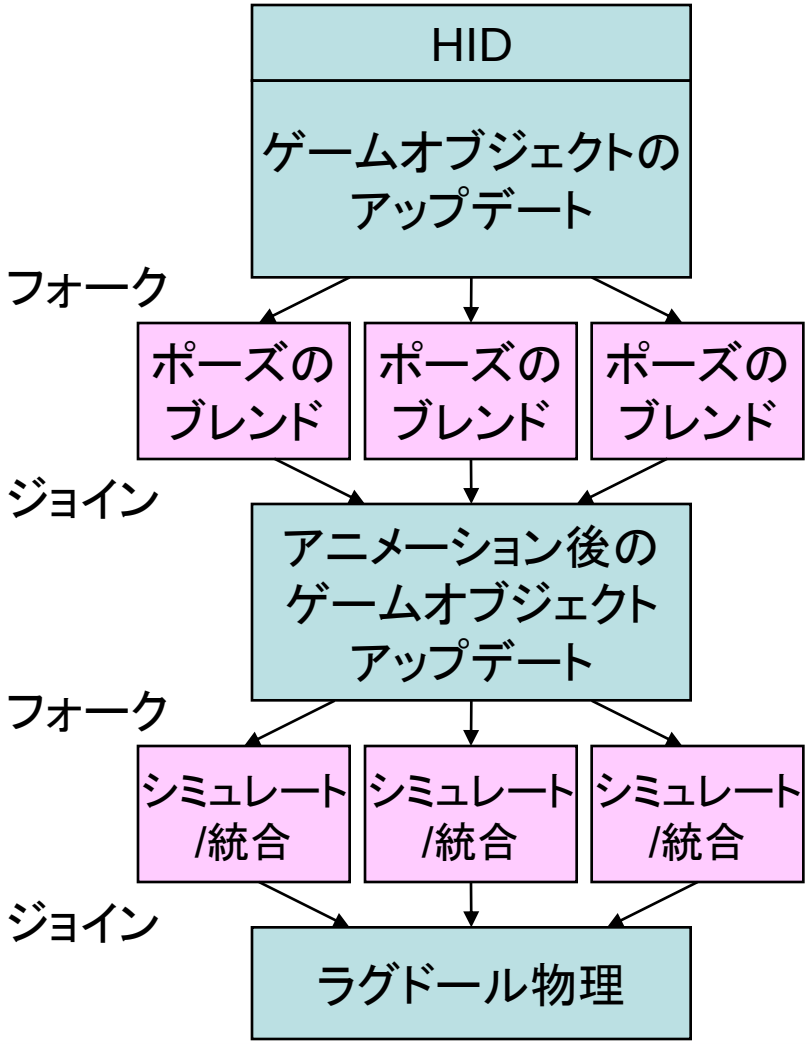
- SIMD
  - 単一の命令で複数のデータを処理する
- フォーク - ジョイン
  - 並列性を持たせるために分割統治アルゴリズムを適用
- サブシステムごとに1つのスレッド
  - 特定のエンジンのサブシステムを別々のスレッドに割り当てて実行
- ジョブ
  - 作業を複数の小さくて比較的独立したジョブに分ける

フォーク - ジョイン

- 並列性を持たせるために分割統治アルゴリズムを適用
  - 粒度が細かな計算は自動化しやすい
    - N/#プロセッサを割り当て
    - 処理内容によって終了時間に差が発生
      - 最も遅いプロセッサに足を引っ張られる

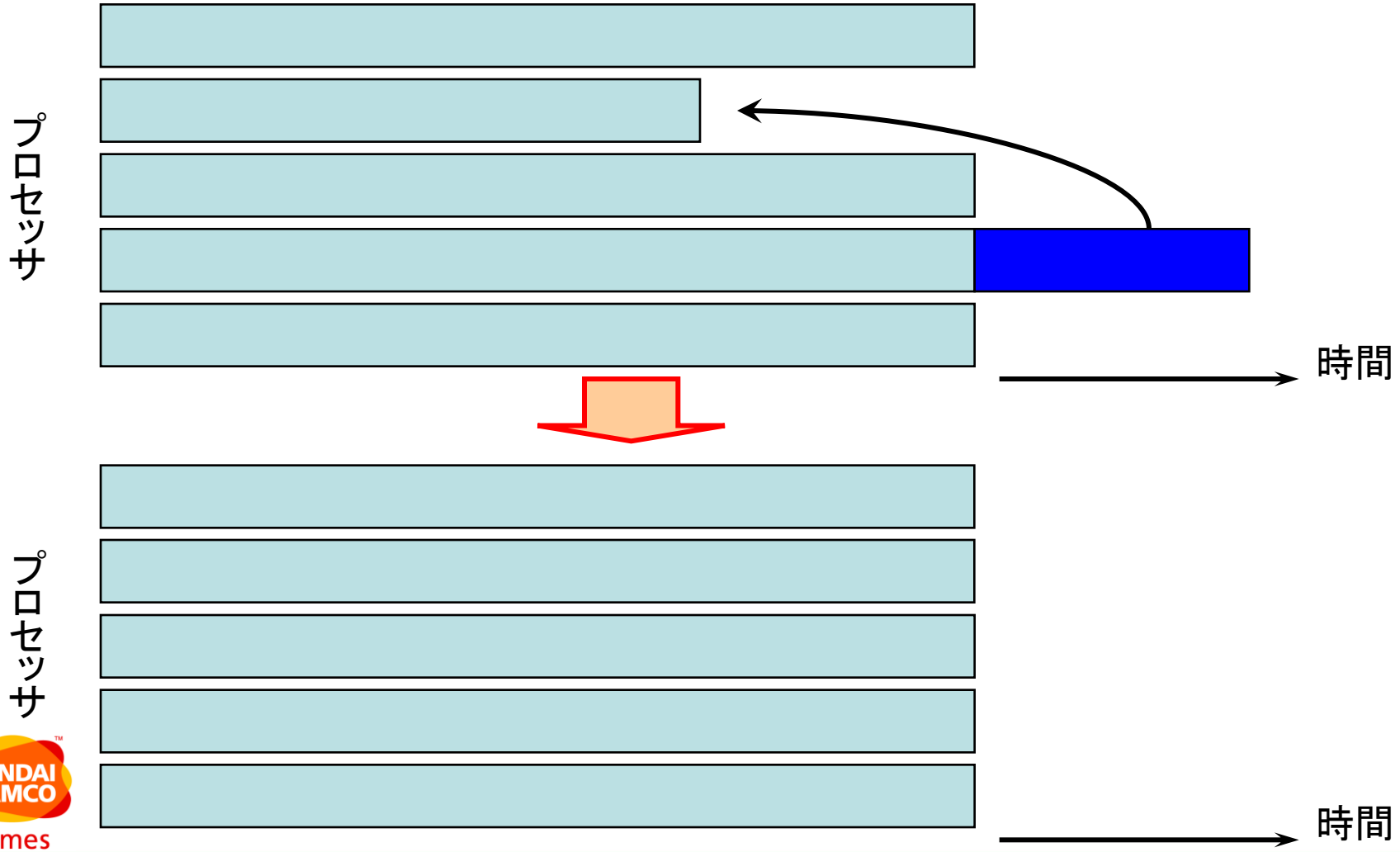


  - コンパイラ任せにできつつある部分



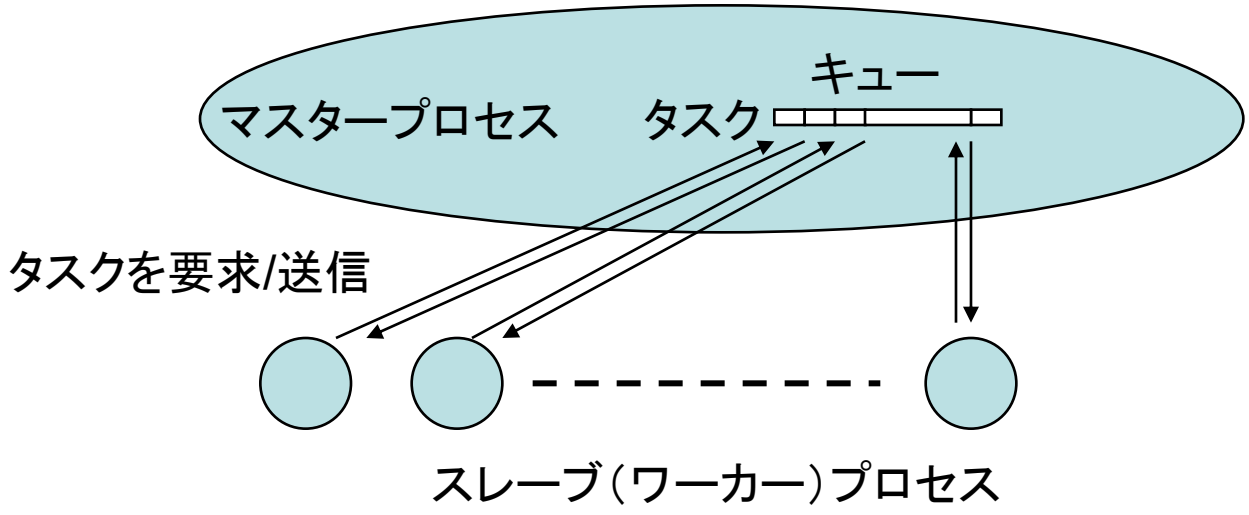
負荷平均化

- 並列化を効率よく行うために



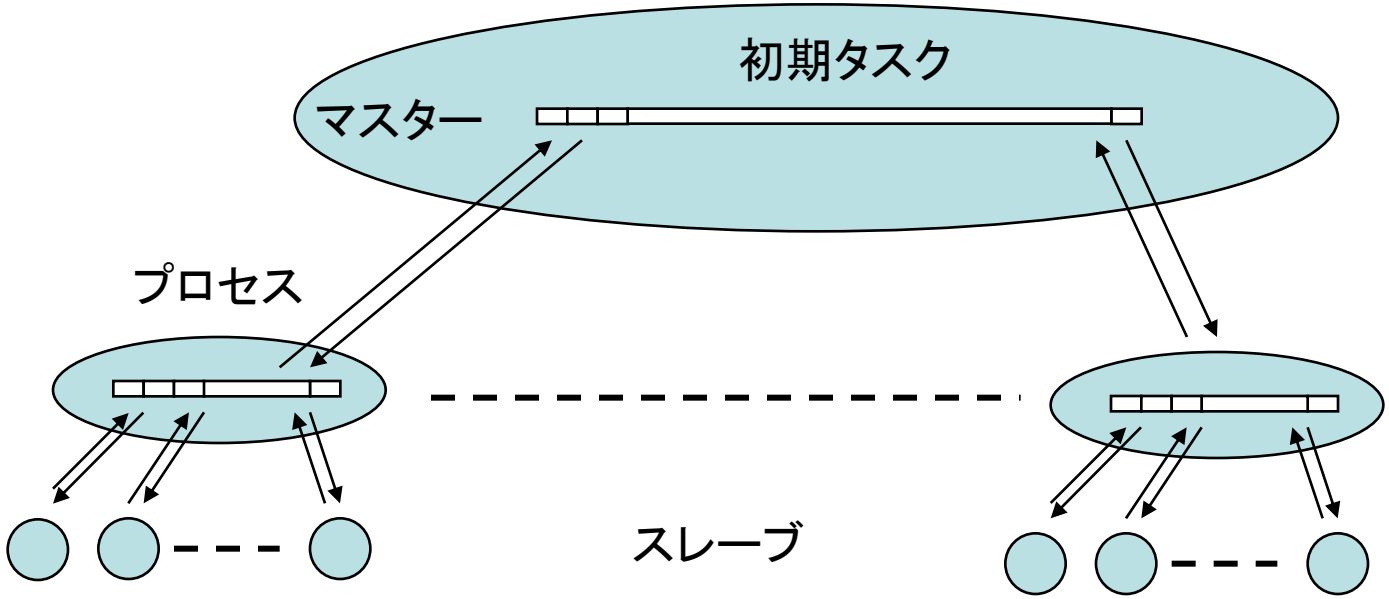
中央管理型ワークプール

- どのような方法で仕事を平等に割り振るか?
- 1つのプロセスが、別のプロセス群からタスクを要求されたら、未処理の物を割り当てる
  - 特別なプロセスが必要
  - 大量のプロセスから同時に要求が発生するとボトルネックに



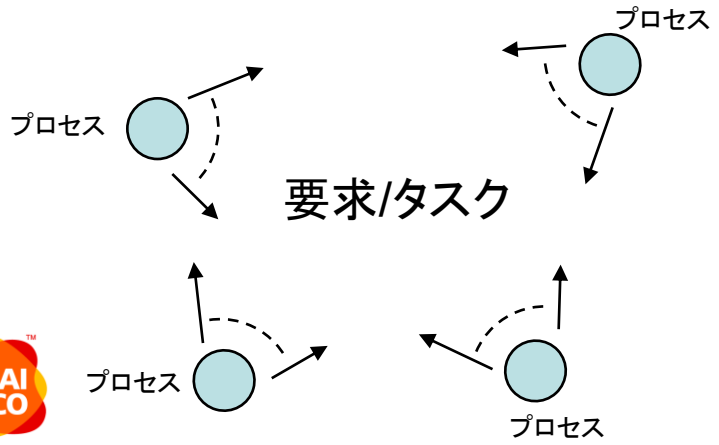
分散型ワークプール

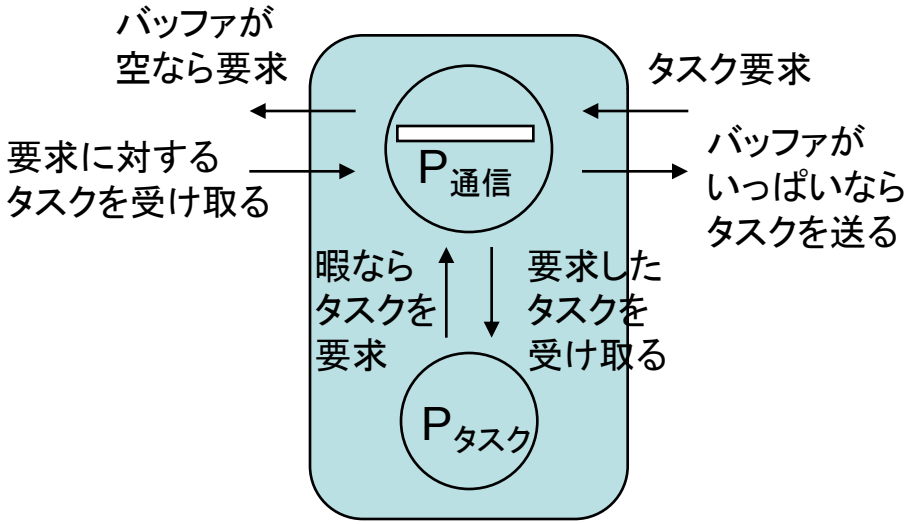
- マスタープロセスの負荷低減
- 階層化し、マスタープロセスの仕事を委譲
  - そもそもタスク管理とスレーブの処理を分けるのはメニーコア時代にどうだろうか？



完全分散型ワークプール

- プロセスが作用しあってタスク群を実行
  - 自分のタスクが無ければ他から取得
  - 要求方法:
    - ラウンドロビン:グローバルカウンタ/#プロセッサ
    - ランダムポーリング:ランダムにリクエスト





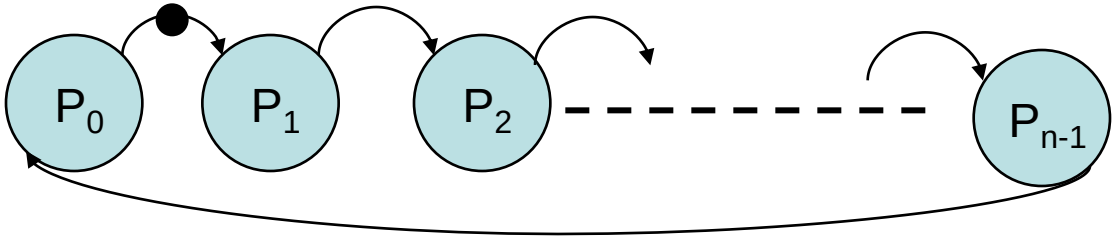
どうすれば全体の処理が終了？

- 独自の終了条件が存在
  - 経路探索等で経路が見つかった！
- 送信中のメッセージが無い
  - モーションの計算を骨ごと別々に計算
    - ??????????

リング終了アルゴリズム

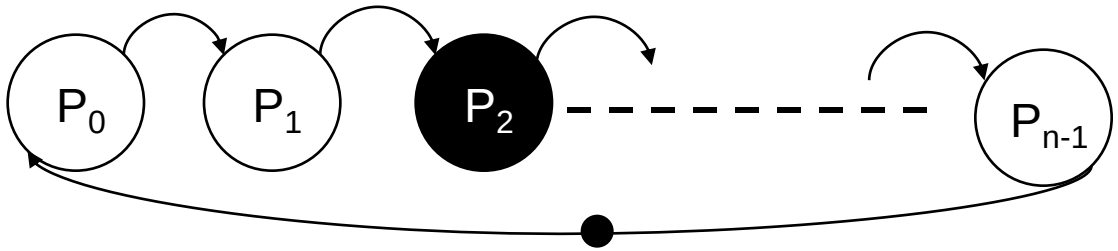
1. 自分の処理が終了したらトークンを受け渡す
2. 最初のプロセスにトークンが戻ってきたら終了

- 他のプロセスからタスクを要求するときは？
  - 一度終わったものが動き出す



前のプロセスに送られるタスク

- P0が終了したらP1に白いトークンを渡す
- 各プロセスが終了したらトークンを次に渡す
- プロセスが、前の(トークンを受け渡しした)プロセスにタスクを送ったらトークンを黒にして受け渡す
- 各プロセスが終了したらトークンの色を変えず次に渡す
- P0が黒いトークンを受け取ったら白いトークンを渡しなおし。白いトークンなら終了

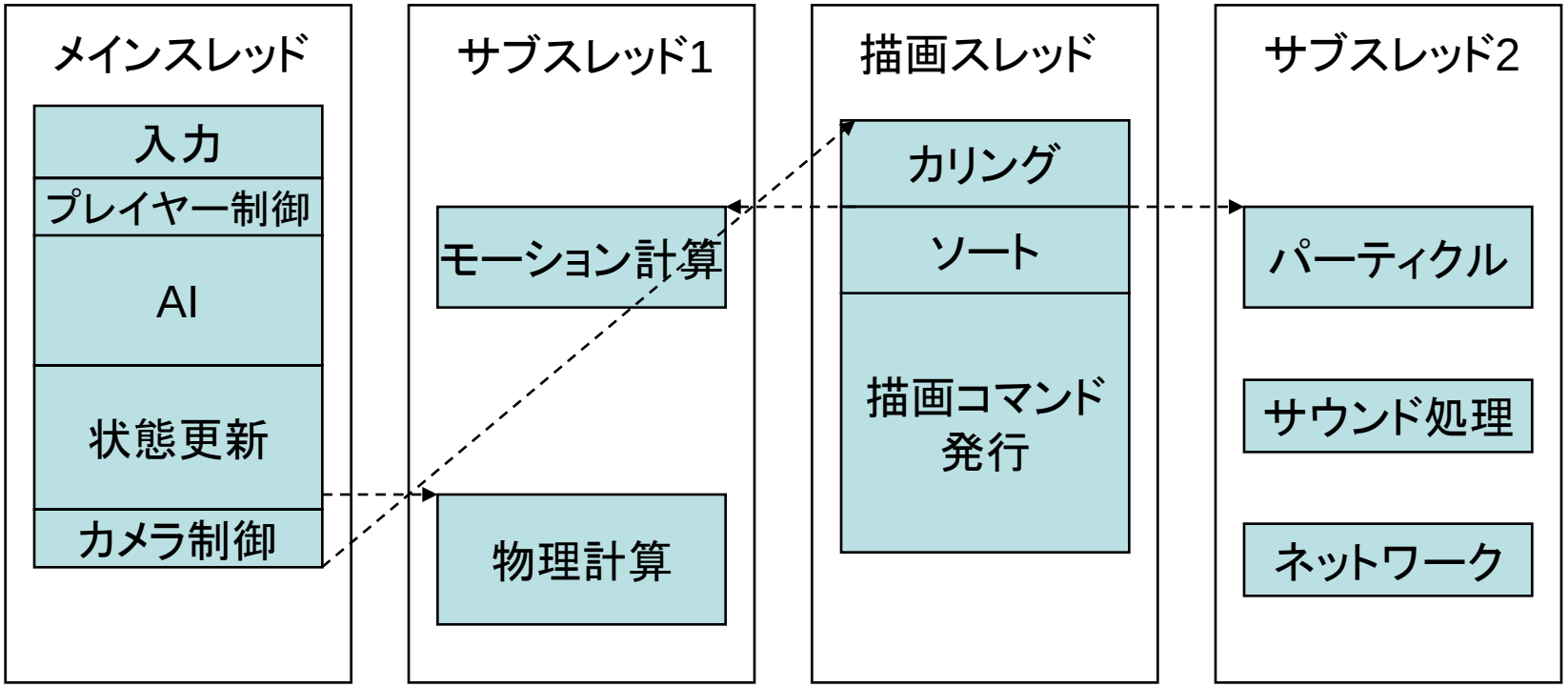


マルチプロセッサのゲームループ

- SIMD
  - 単一の命令で複数のデータを処理する
- フォーク - ジョイン
  - 並列性を持たせるために分割統治アルゴリズムを適用
- サブシステムごとに1つのスレッド
  - 特定のエンジンのサブシステムを別々のスレッドに割り当てて実行
- ジョブ
  - 作業を複数の小さくて比較的独立したジョブに分ける

ゲームエンジンの設計

- 現代的な並列ゲームエンジン



マルチプロセッサのゲームループ

- SIMD
  - 単一の命令で複数のデータを処理する
- フォーク - ジョイン
  - 並列性を持たせるために分割統治アルゴリズムを適用
- サブシステムごとに1つのスレッド
  - 特定のエンジンのサブシステムを別々のスレッドに割り当てて実行
- ジョブ
  - 作業を複数の小さくて比較的独立したジョブに分ける

BANDAI NAMCO Games

ゲームエンジンの設計

- これからのゲームエンジンアーキテクチャ
  - コアの多能工化
  - とにかく細粒化
    - スケーラビリティを確保するためには、オーバーヘッドが存在しても隙間を空けないようなシステムに

細粒化

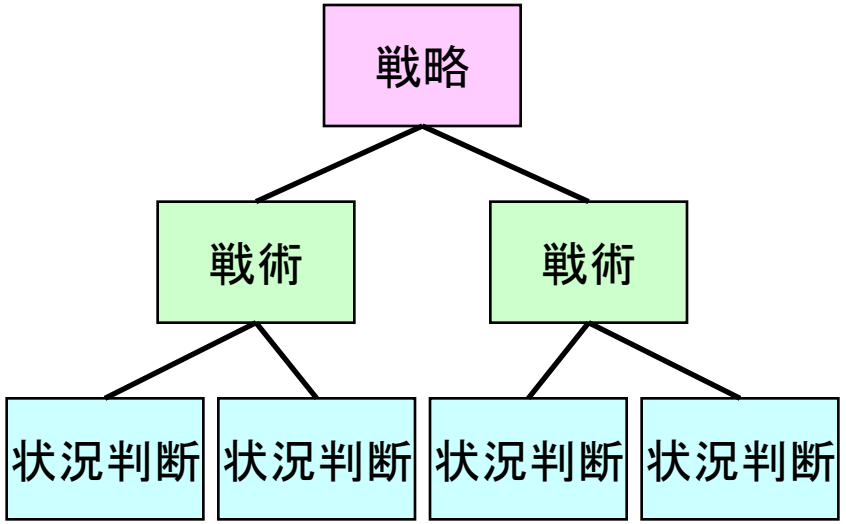
- 今までのゲームを細かくするのは限界
  - グローバルイルミネーションエンジンへの移行?
  - パーティクルの増大
  - 流体シミュレーションの導入
  - 物理エンジンの要素の増大(含む破壊)
- ゲーム自体を並列化向きにデザインする
  - 多くのキャラクター
    - アニメーション
    - AI
  - 巨大都市
    - プロシージャルコンテンツ生成
  - カメラを入力装置とし、詳細に動画を分析

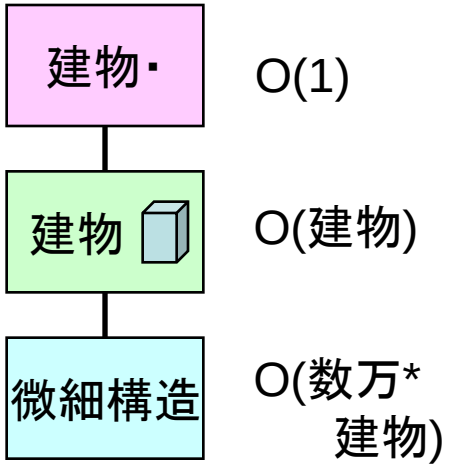
目次

- はじめに
- メニーコア化
- マルチプロセッサのゲームループ
- メニーコア ゲームデザイン
- バスの帯域問題

並列化向きデザイン

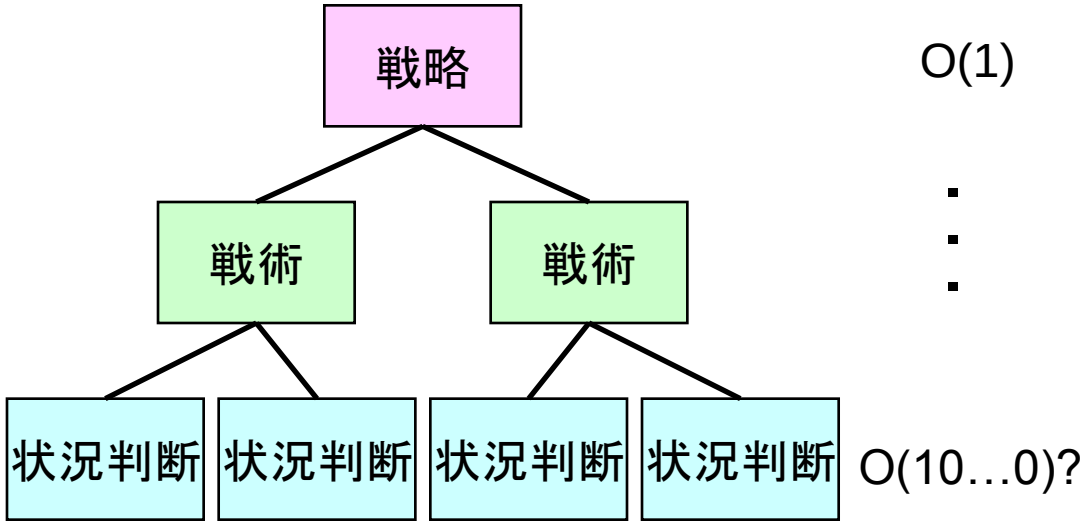
- 多くのキャラクター(AI)
  - 全体：戦略
  - 部隊：戦術
  - 個人：状況判断
- 巨大都市
  - 遠：航空写真のような分布情報
  - 中：バリエーションを持つ街並み
  - 近：細かなディテールを持つ建物





どこまでいける?

- データサイズから
  - ID(4 bytes)
  - 位置(12 bytes)
  - 速度(12 bytes)
  - 状態(4 bytes?)
- = 32 bytes
- 1GBとすると、3355万体
  - 非常にマッシブなゲーム



1000万体のキャラクターを扱えるのか?

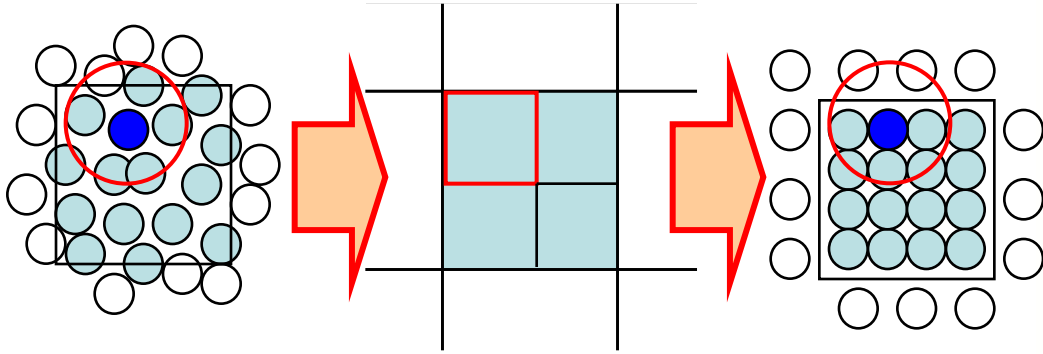
- 最新のGPUでも10億ポリゴン/秒
  - 点ぐらいなら今でもできる(たぶん?)
- フルHDで全体にキャラクタを(非常に遠方に)ばら撒いたとすると10体/ピクセル
  - 点以下の情報
  - 個別で扱う価値はあるのか?
  - おそらく密度分布で扱う必要がある

密度分布?

- 流体シミュレーション
  - 粒子法
    - 粒子の運動に離散化してシミュレーション
  - 格子法
    - 空間を碁盤の目のように格子に切り分けシミュレーション
- CEDEC 2009:
  - 破壊シミュレーション入門

相互作用範囲をベースにした計算精度の決定

- 相互作用する距離の精度での近似計算を用いることで、挙動を再現する再構築が行われる
  - テイラー展開
  - 球面調和関数展開
  - Wavelet 展開



- 流体シミュレーション等だけでない

Boid
視野に入る領域

流れ弾

Walther HP. Made by Spreewerke (cyq) in early 1944. http://www.adamsguns.com/

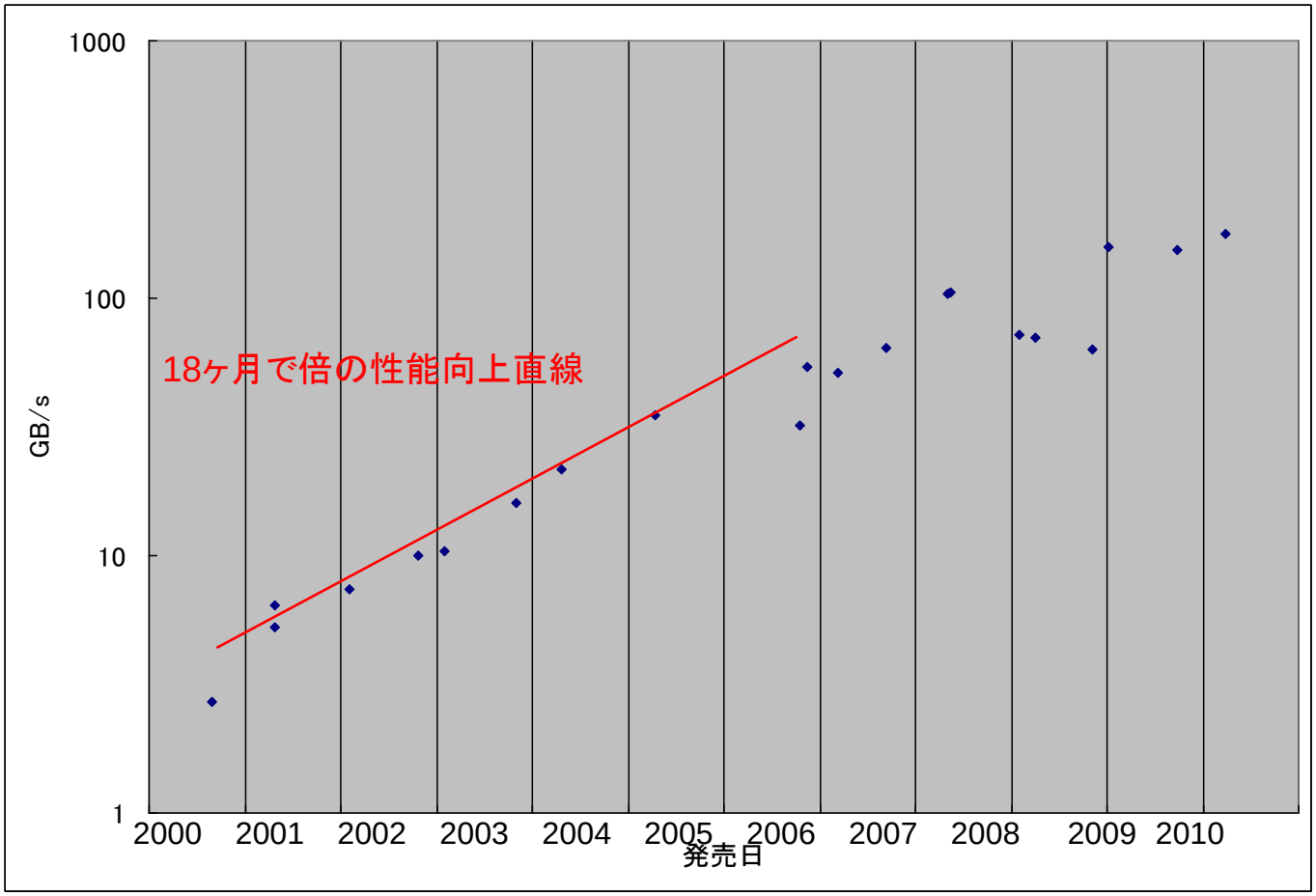
目次

- はじめに
- メニーコア化
- マルチプロセッサのゲームループ
- メニーコア ゲームデザイン
- バスの帯域問題

何が問題

- バスの速度が追いつかない
  - 増加速度は減少

主なビデオカードにおけるバス幅

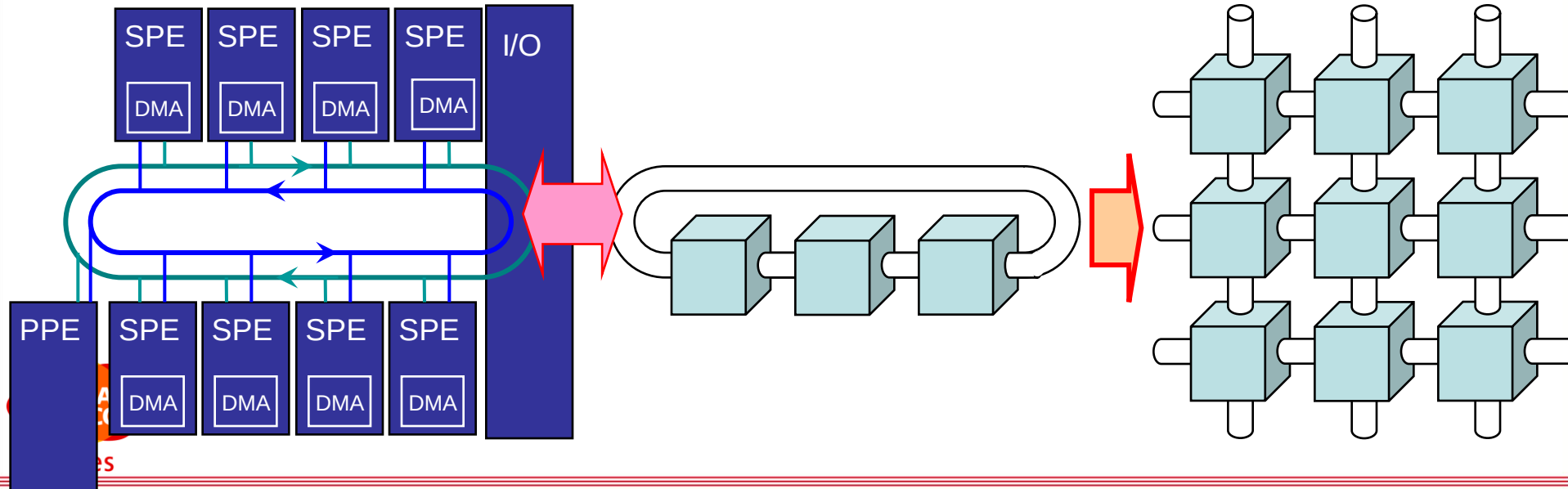


データの転送が追いつかないならどうすればよいか

- CPU⇔メモリ間の転送を減らす
  - 超巨大キャッシュ
    - 現代でさえ、PPEと同じ程度のフットプリントをL2キャッシュに使っているのに…
  - メモリを使わないで計算を行う

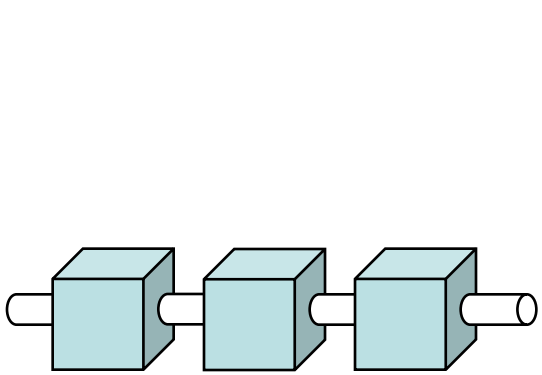
メモリを使わない計算

- メモリバスの方向性
  - 全てのチップが1つのメモリハブを経由してデータを受け渡すのはムダ
  - 既にSPE-SPE間の転送のようにメモリを介さないデータの受け渡しは我々の手の中にある
  - 多次元化

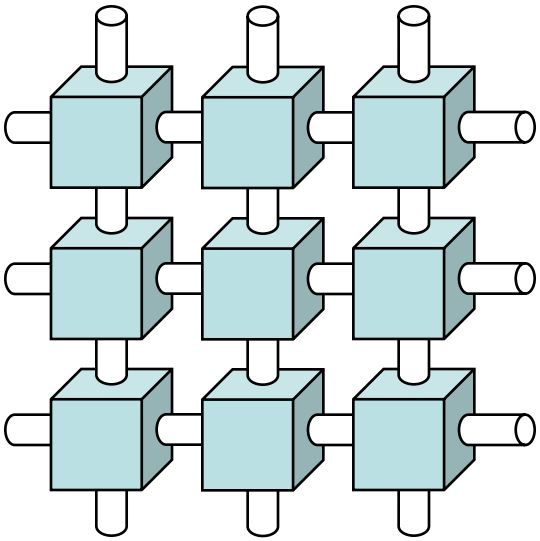


次元とアルゴリズム

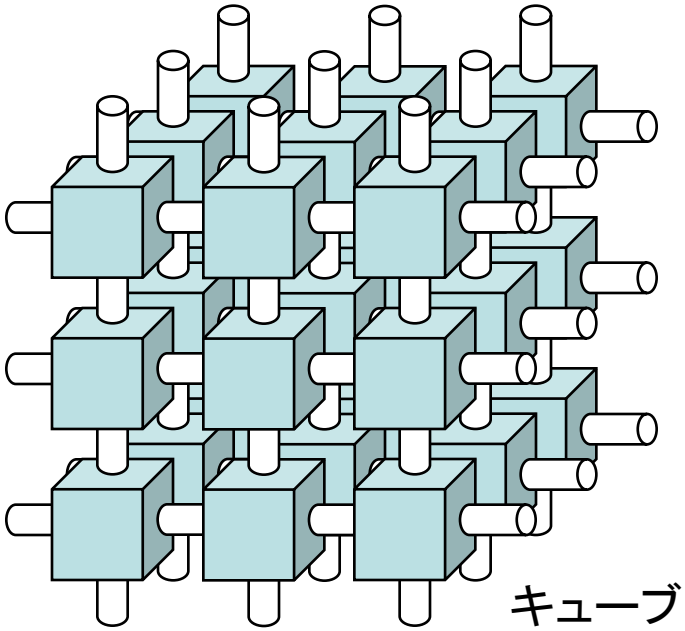
- ダイ間の結合網はアルゴリズムに影響
  - パイプライン作業はリング構造が
  - 行列的な作業はメッシュが
  - 立体的な構造を考えるのは、キューブか？
    - 次世代CPUは、立体チップ?



リング

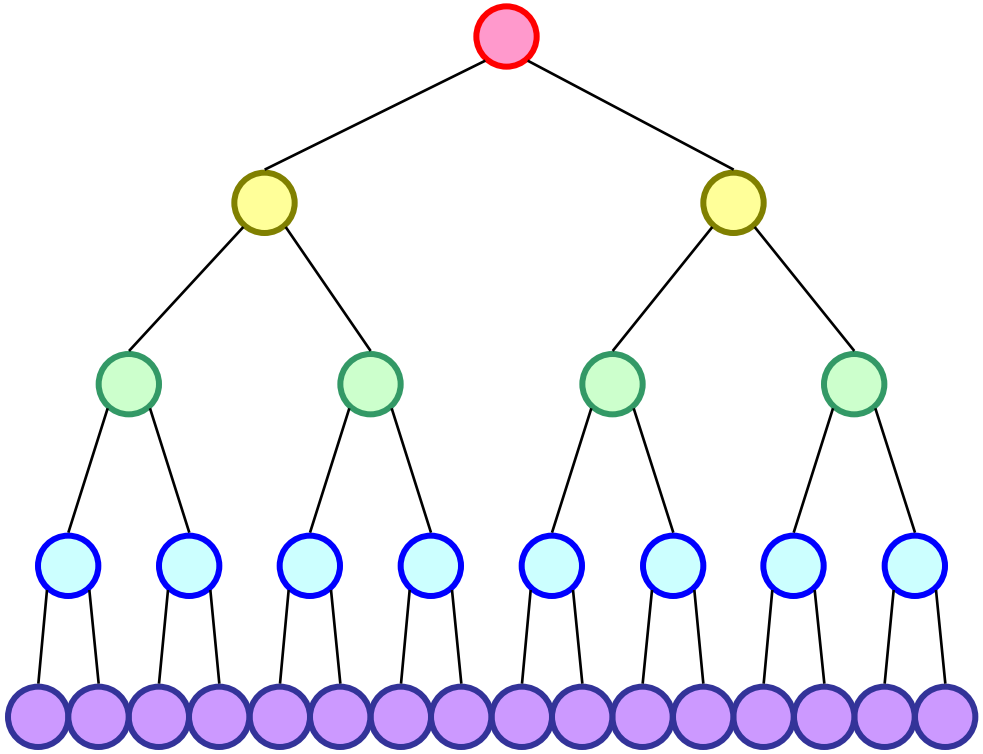


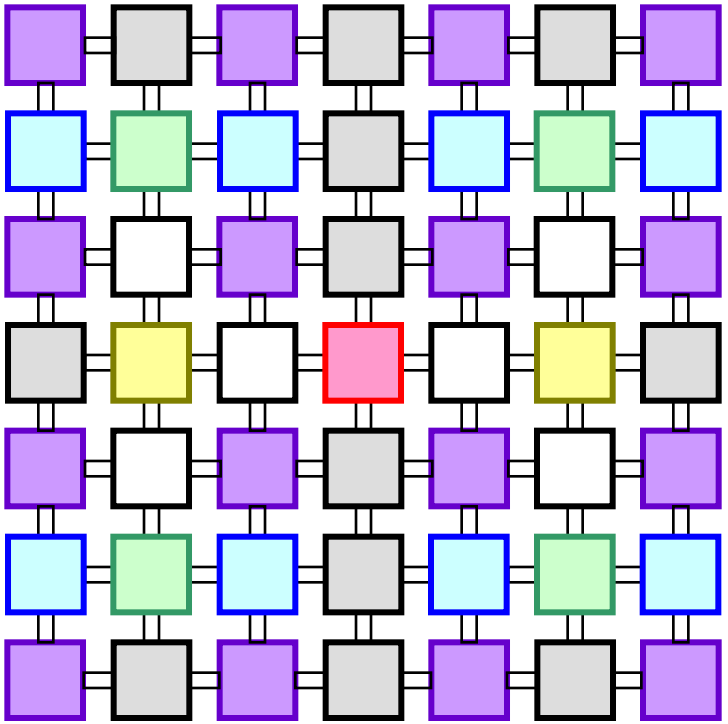
メッシュ



キューブ

ツリーのメッシュへの埋め込み





まとめ

Many-core, the final frontier

- 据え置き型系の技術で、まだ遅れていない分野

- アルゴリズムのパラダイムシフトが必要
  - 高局所性
  - 細粒度
  - 階層化戦略
  - コア、バスの明示的な指定（必要なら実装されるでしょう）
- 技術がゲームデザインに影響しても良い
  - 技術シーズの製品の成功は、難しいが、否定することはない

BANDAI NAMCO Games