ネットワークゲームの仕組みとゲームデザイン

株式会社セガ
第三CS研究開発部
節政暁生

目的

- ネットワークゲームを作る上での制約を知ってもらう。
- 色々なゲームの通信方式と特徴を覚えてもらう。
- ゲームデザインする上で何に気を付けるべきか考える。
- ネットワークゲームの技術的な部分は2000年ぐらいまでで大体出そろっている。今後の応用にも役立つはず。

概要

- ネットワーク技術のおさらい
- 完全同期型ゲームの例
  - 格闘ゲーム
  - RTS
- 非同期型ゲームの例
  - FPS
  - PSO、PSU
- まとめ

遅延と帯域とパケットロス

ネットワークはLAN (Local Area Network) と WAN (Wide Area Network) の2種類に分けられる。LANとWANでは環境の条件が全然変わってしまう。

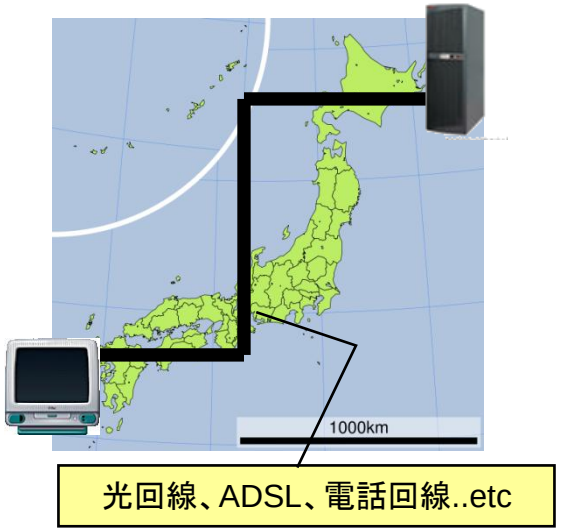
物理的に近い距離の接続 ＝ LAN

Ex.会社内での接続

遅延: 1～2 ms以内
帯域: 1Gbps
パケットロス: ほとんど無し

物理的に遠い距離の接続 ＝ WAN

Ex.インターネット全般

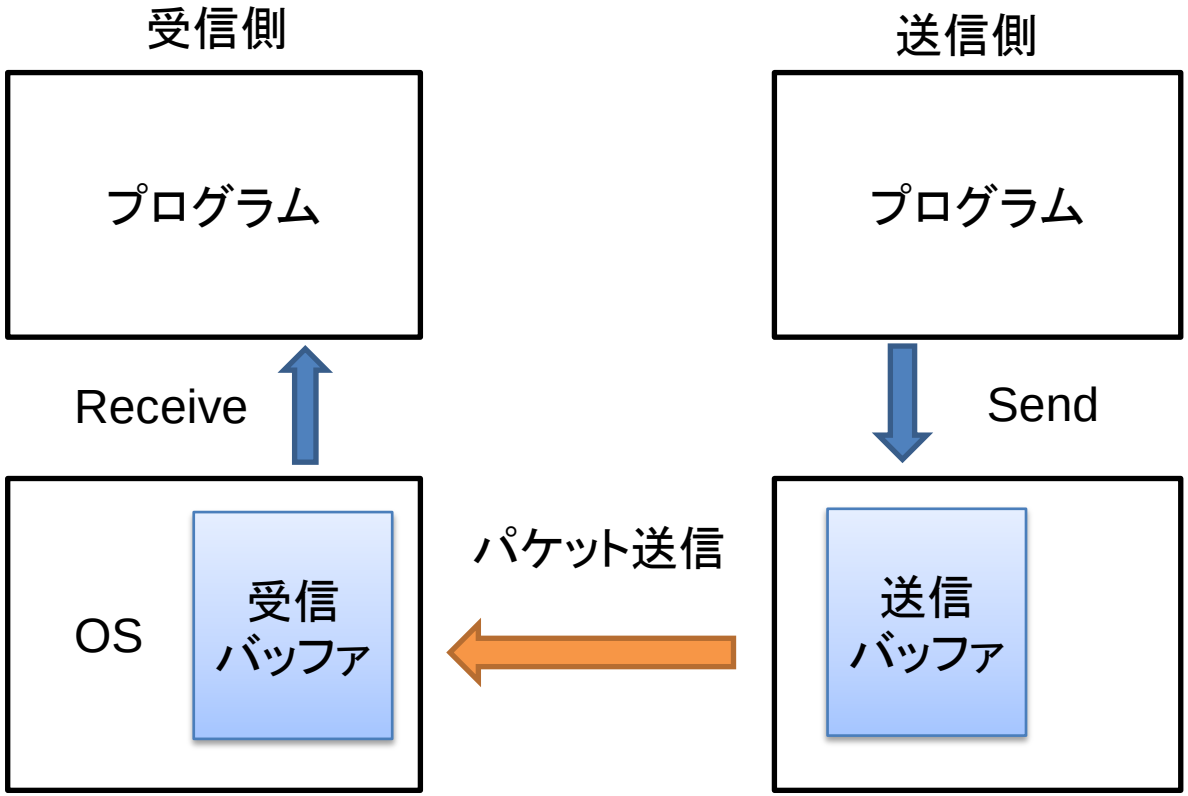


遅延: 数ms～数百ms
帯域: 28kbps～50Mbps程度
パケットロス: 環境による

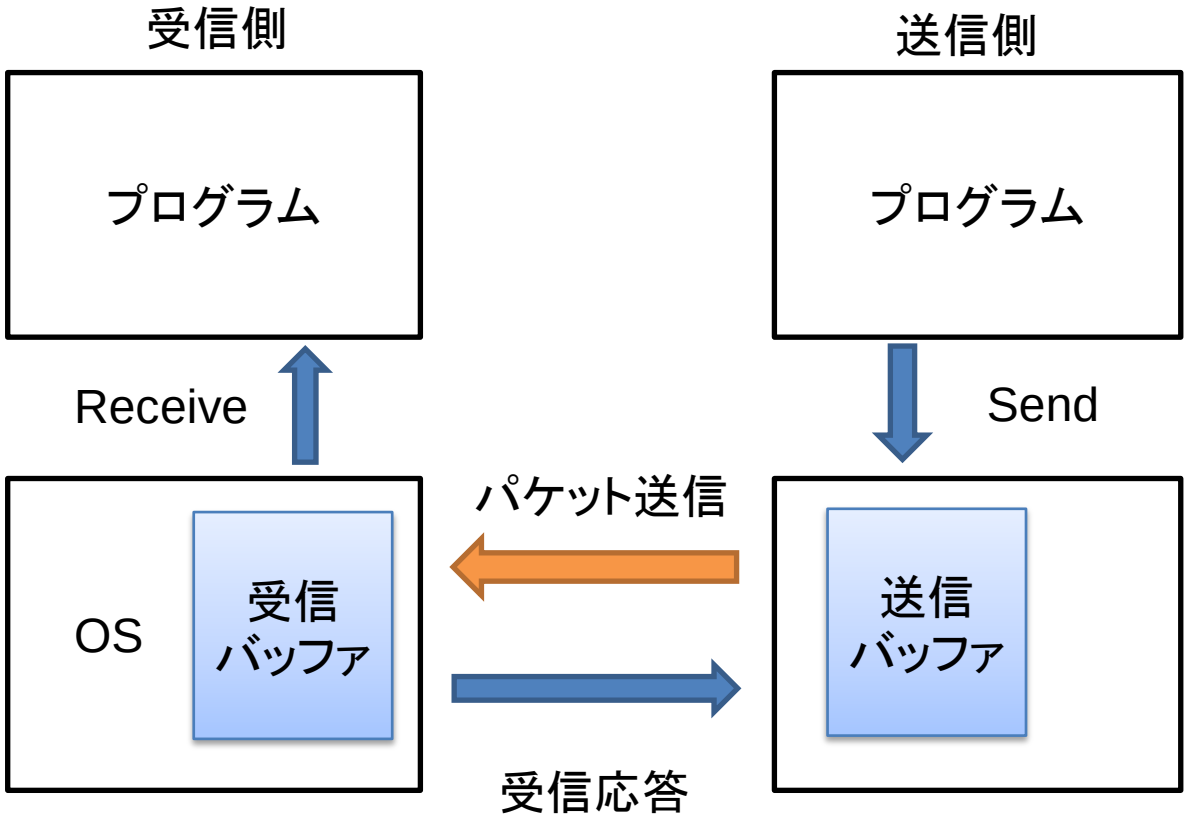
通信の種類

UDP	TCP	Reliable UDP
パケットが届かない場合がある	パケットは必ず届く	パケットは必ず届く
パケットの順番が変る場合がある	パケットは順番が保証される	パケットの順番は実装次第
ヘッダサイズが小さい IPヘッダ込みで28バイト	ヘッダサイズが大きい IPヘッダ込みで40バイト	ヘッダサイズは実装による
処理が速い	処理が遅い	処理が速い
NAT問題が発生しやすい	NAT問題が発生しにくい	NAT問題が発生しやすい

UDPの簡単な説明



TCPの簡単な説明

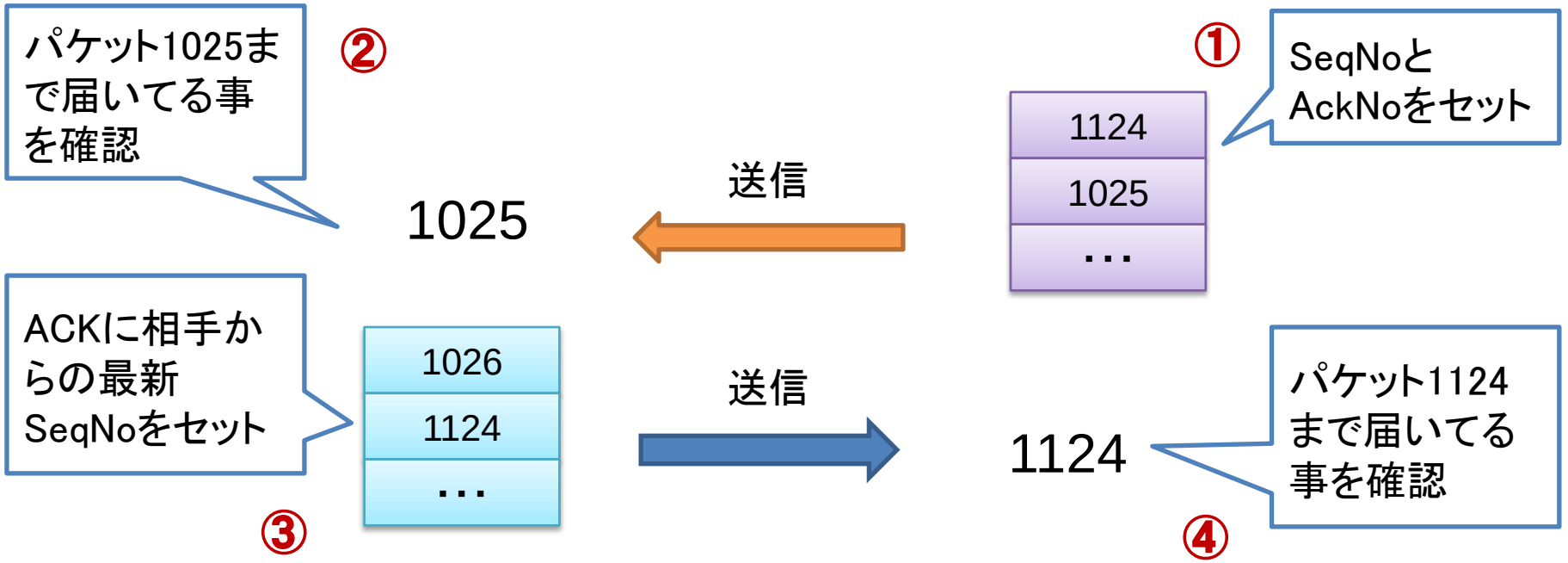

受信側
送信側
プログラム
プログラム
Receive
Send
パケット送信
OS
受信
バッファ
送信
バッファ
受信応答

RUDPの作成例 ① 受信応答のしくみ

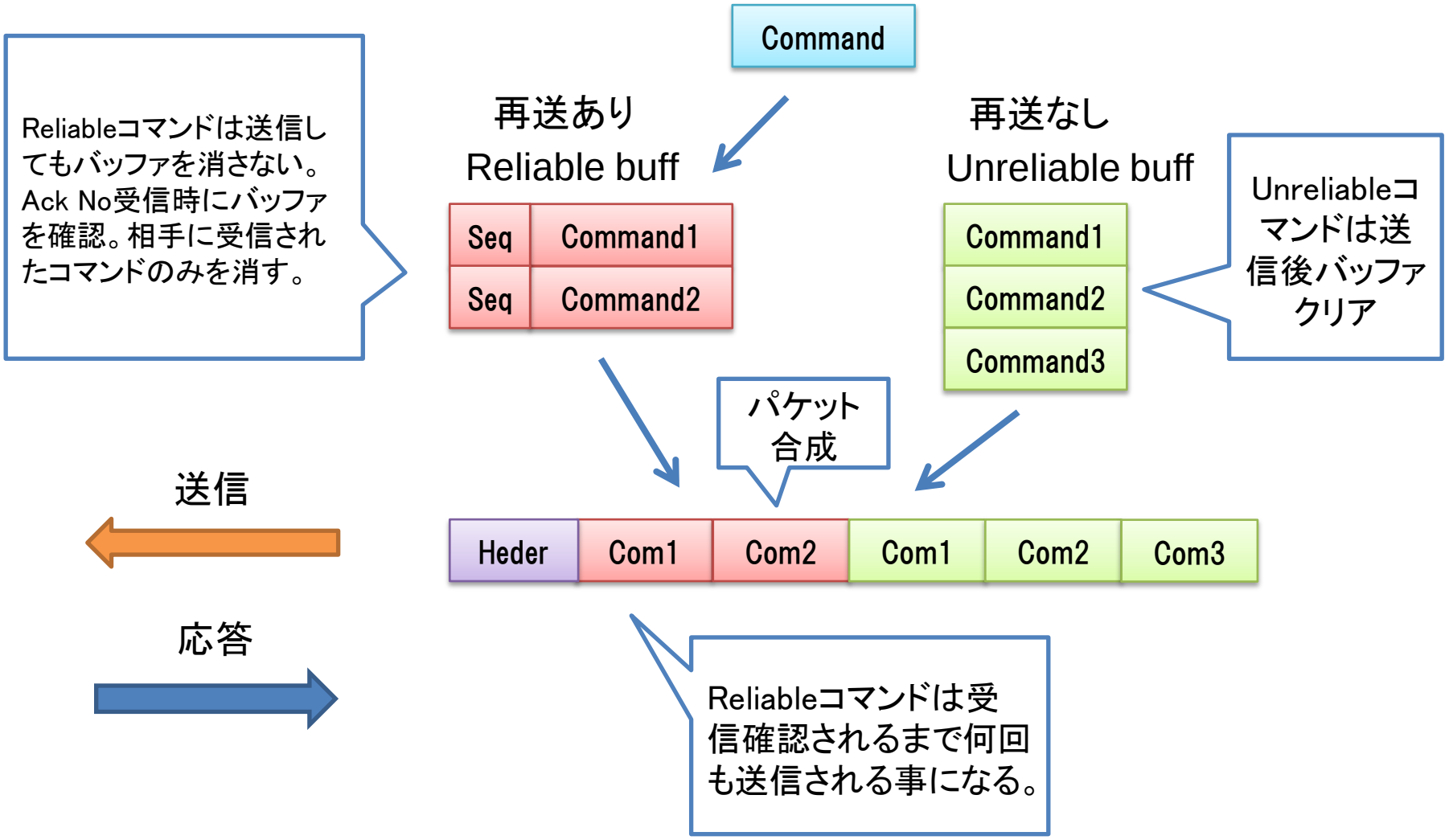
パケットヘッダ

SEQ No
ACK No
Command

- シーケンスナンバーはパケットを送信する度に+1される数値。
- 自分がパケットを送信する際は相手から受け取ったパケットのシーケンスナンバーをACK No に入れて返すようにする。
- 定期的にお互いパケットを送信していれば、自分のパケットが何番目まで届いているかが解る。



RUDPの作成例 ② データー再送のしくみ



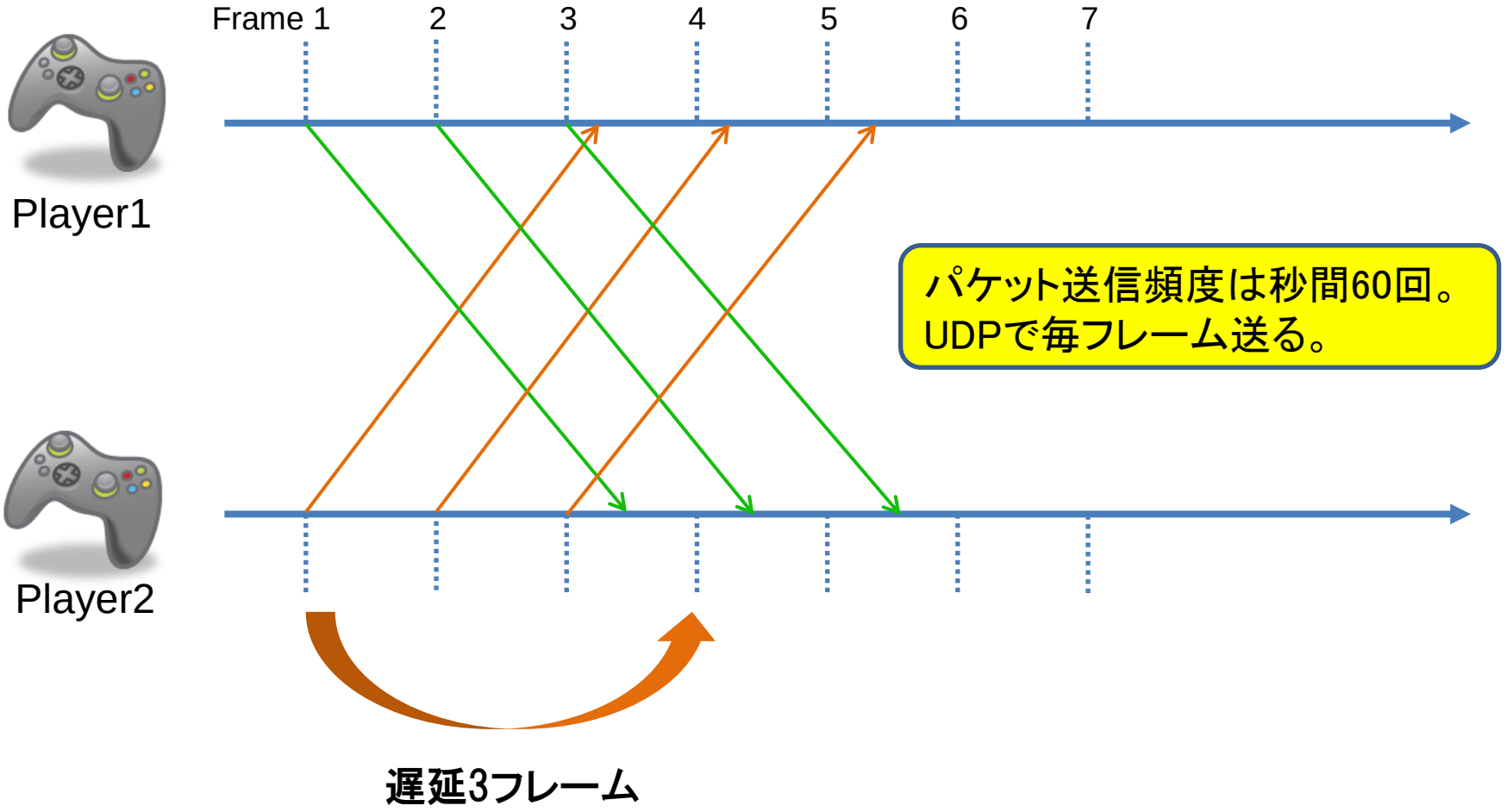
格闘ゲームの場合

Virtua Fighter 5

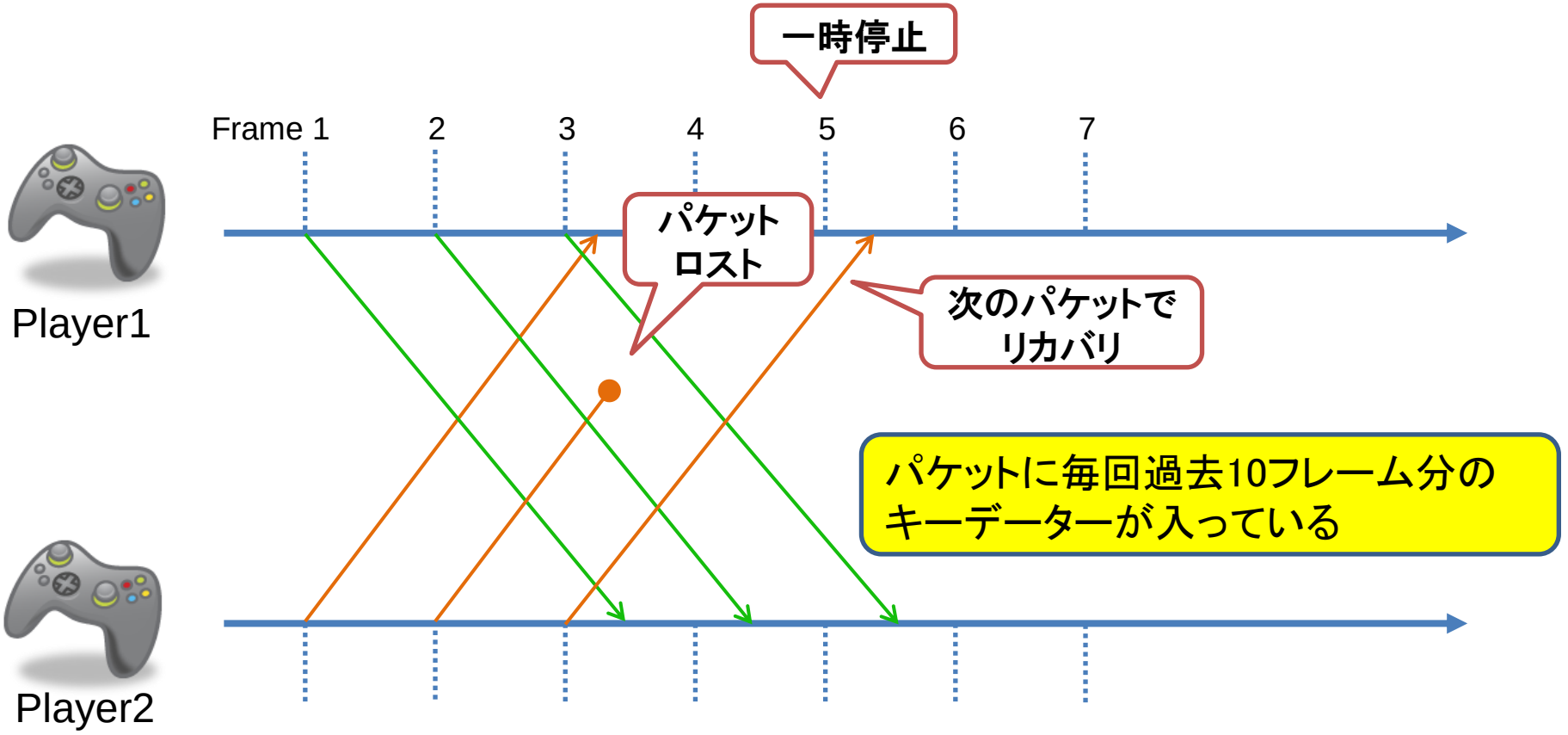
バーチャファイターの要素

- 二人で戦う。
- 1/60で動作。
- アクション性が非常に高い。
- 入力は方向キー+3ボタン。
- 試合時間は割と短め。
- 試合中にロード等は入らない。

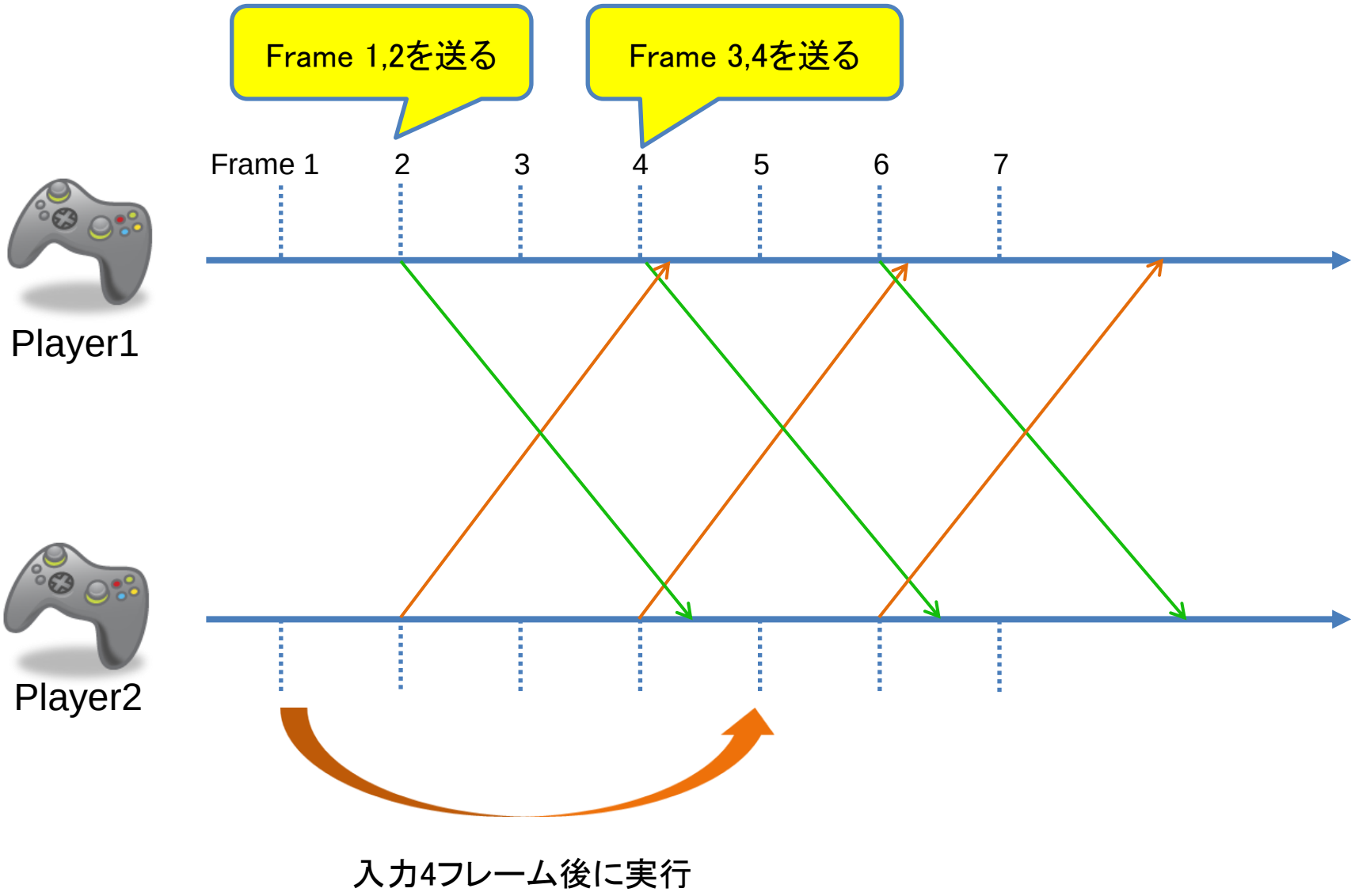
キー入力同期方式


Frame 1
2
3
4
5
6
7
Player1
Player2
パケット送信頻度は秒間60回。
UDPで毎フレーム送る。
遅延3フレーム

エラー訂正の工夫


一時停止
Frame 1
2
3
4
5
6
7
パケット
ロスト
次のパケットで
リカバリ
パケットに毎回過去10フレーム分の
キーデーターが入っている
Player1
Player2

パケットの頻度を落とす場合



RTSの場合

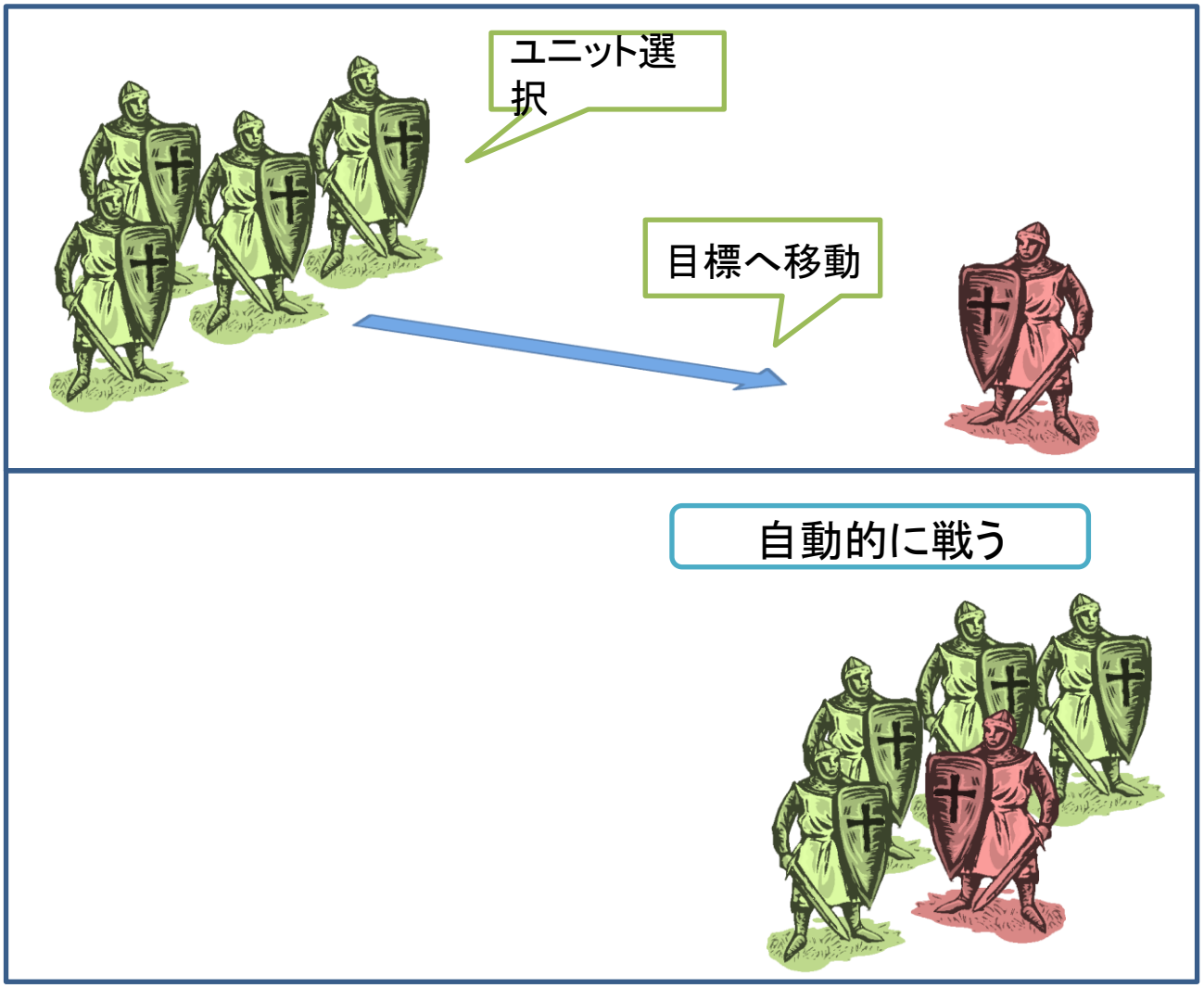
Age of Empires
(1997)

Starcraft
(1998)

RTSの要素

- 最大8人まで接続可能なマルチプレイ。
- プレイヤー以外にCPUと戦うことも出来る。
- 各自数百体のユニットを同時に操作する。
- 戦闘の結果は全て同期してなければならない。
- モデムでも動く程度の通信帯域。

RTSの操作コマンド



コマンド入力同期方式

```
Select   probe
Build   x, y, Gateway

Select  Gateway
Train  Zealot
Train  Zealot

Select  unit01, 02,  03, 04
Move  x, y, instant
Move  x, y, queue
Move  x, y,  queue

Attack Move  x, y,  instant
```

基本的にはマウスで操作した
コマンドがそのまま送られている
イメージ。

「選択しているユニット」に対して
命令が実行されるようになっている。

ユニット1つ1つにコマンドを送るわけ
ではないので各コマンドはわずか
数バイト。

細かく考えればターン制

- 1秒間に何回もターンがあると考えると解りやすい。
- 1回のターンに何個もコマンド入力出来る。
- 何もしなくてもターン終了。
- 通信状況が悪かったら1秒間当たりのターン数を減らせばよい。

もうちょっと単純な例

チューチューロケット
1999年

プレイヤーは「矢印パネル」を置くだけ。

ネズミのAI
・何も無い所はまっすぐ
・パネルの上では矢印の方向へ曲がる
・壁にぶつかったら右へ

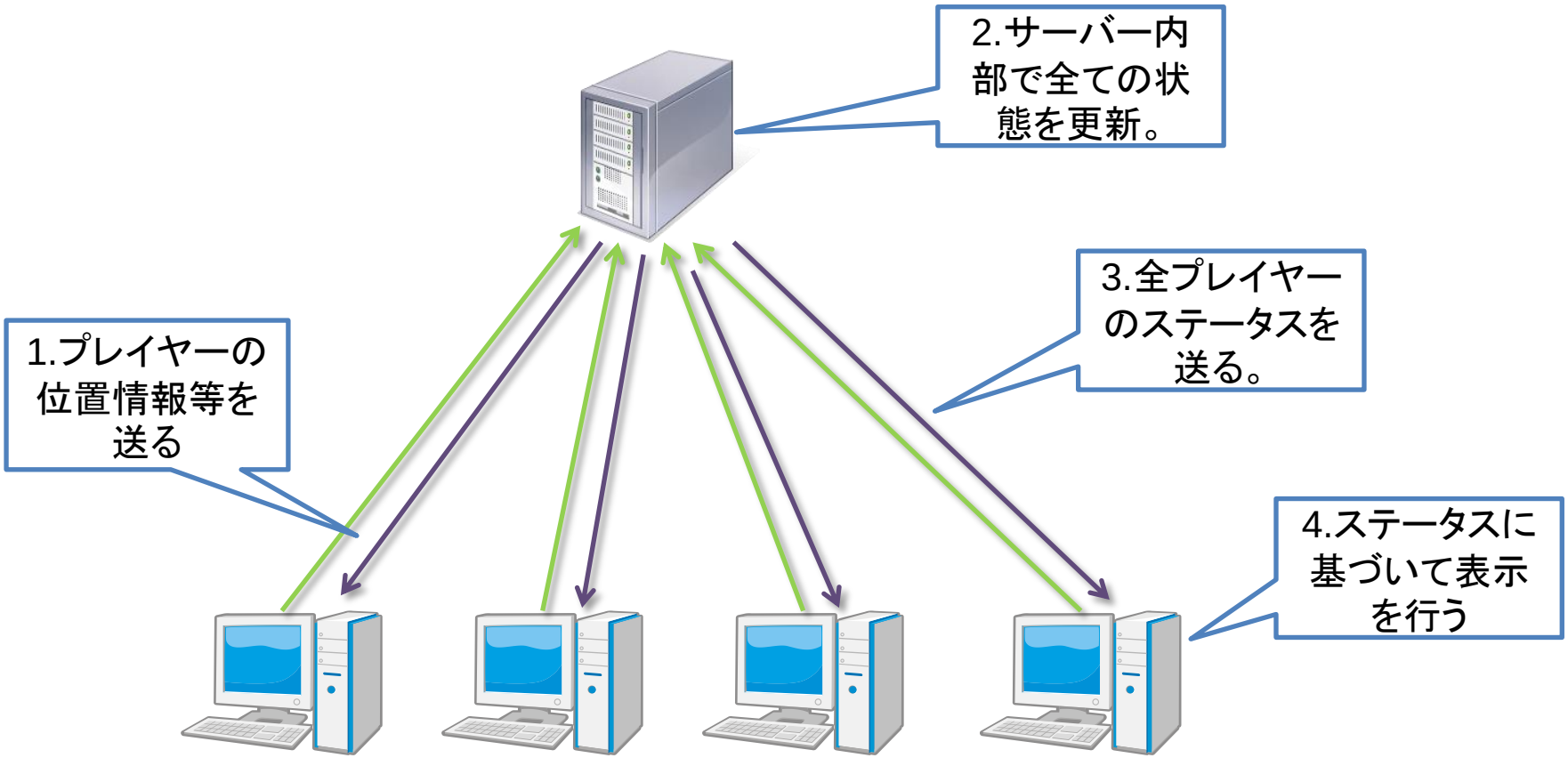
非同期型ゲーム

- 完全同期型の欠点
  - ラグが気になる
  - 多人数に対応し辛い
  - 回線品質に左右される
  - フレームレート固定
- 非同期型では
  - 同期待ちしない。ゲーム状態のずれを許容する
  - 同期したいステータスを全て送る必要がある
  - プログラム的には複雑な作りになる
  - どの要素を捨てるかがゲームデザインに関わる

FPSの場合

Quake3
1999年

サーバー集中処理型


2.サーバー内部で全ての状態を更新。
1.プレイヤーの位置情報等を送る
3.全プレイヤーのステータスを送る。
4.ステータスに基づいて表示を行う

ステータス情報

```
typedef struct entityState_s {
    int        number;    // entity index
    int        eType;     // entityType_t
    int        eFlags;
    trajectory_t    pos;  // for calculating position
    trajectory_t    apos; // for calculating angles
    int        time;
    int        time2;
    vec3_t     origin;
    vec3_t     origin2;
    vec3_t     angles;
    vec3_t     angles2;
    int        otherEntityNum;   // shotgun sources, etc
    int        otherEntityNum2;
    int        groundEntityNum;  // -1 = in air
    int        constantLight;    // r + (g<<8) + (b<<16) + (intensity<<24)
    int        loopSound;        // constantly loop this sound
    int        modelindex;
    int        modelindex2;
    int        clientNum;        // 0 to (MAX_CLIENTS - 1), for players and
corpses
    int        frame;
    int        solid;            // for client side prediction, trap_linkentity sets
this properly
    int        event;            // impulse events -- muzzle flashes, footsteps, etc
    int        eventParm;
    // for players
    int        powerups;         // bit flags
    int        weapon;           // determines weapon and flash model, etc
    int        legsAnim;         // mask off ANIM_TOGGLEBIT
    int        torsoAnim;        // mask off ANIM_TOGGLEBIT
    int        generic1;
} entityState_t;
```

プレイヤー等の状態を表すデーター
ポジション、角度、その他、色々

1体あたり164byteくらい。
これを人数分定期的送っている。
(秒間20回)

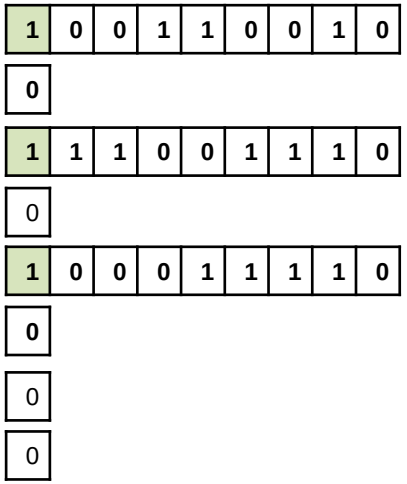
例えば8人で対戦するとなるとかなり
データーが大きくなる。

デルタ圧縮

	Last	New	delta
Pos X	100	150	50
Pos Y	10	10	0
Pos Z	-10	-60	-50
Angle X	20	20	0
Angle Y	60	90	30
Angle Z	0	0	0
Animation	3	3	0
Action	1	1	0

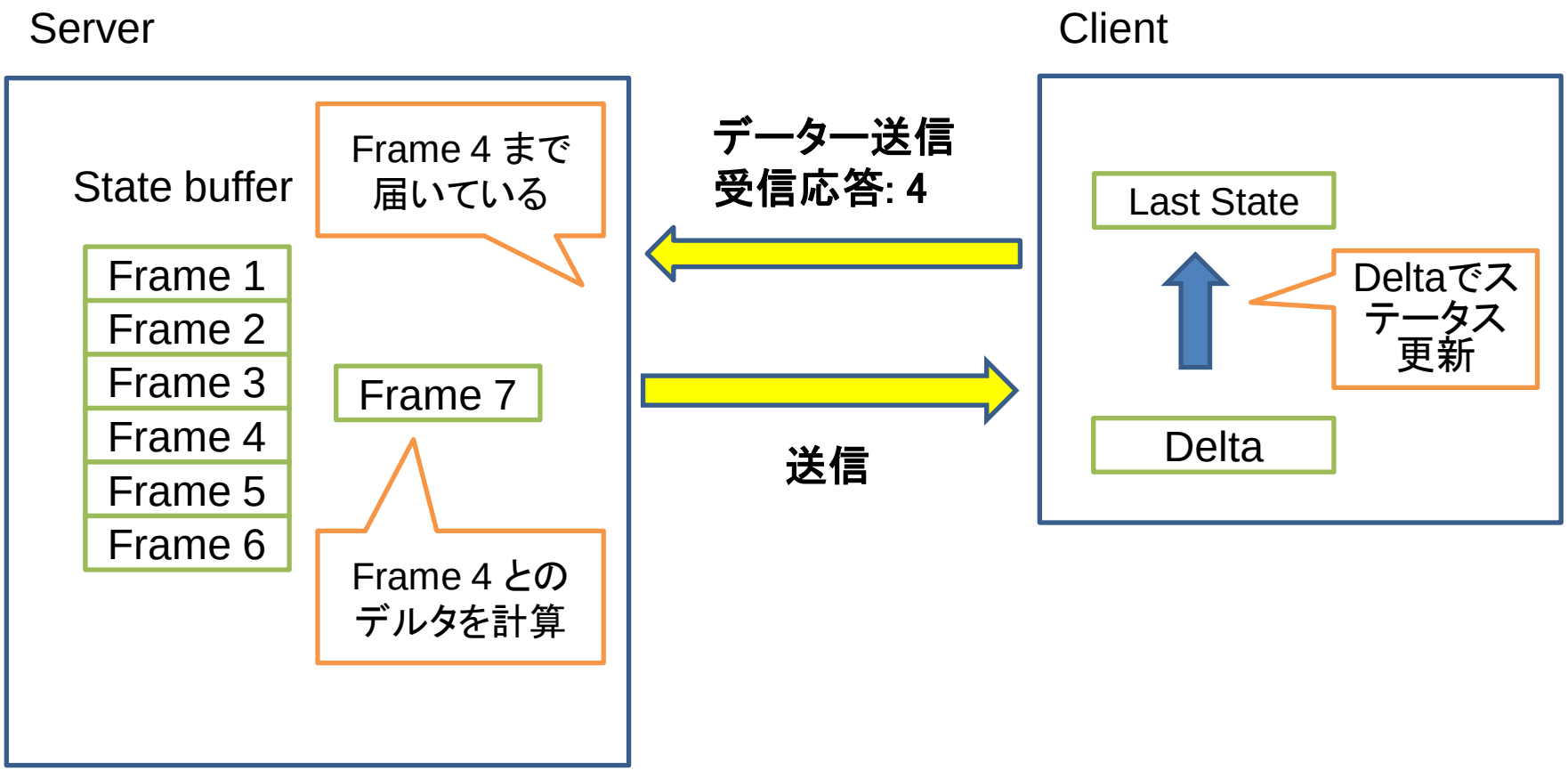
差分

Bit stream 化



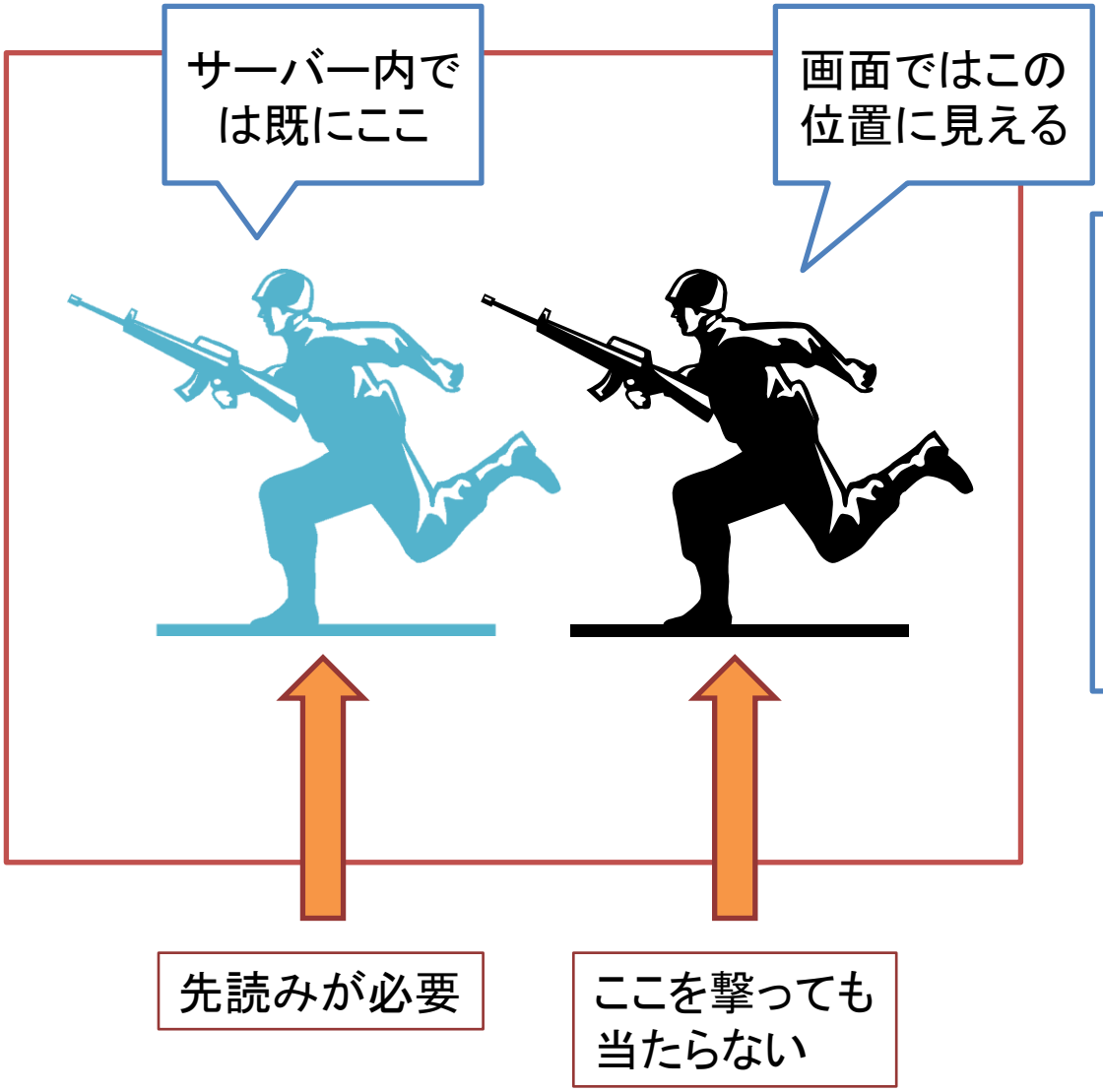
- ステータスのパラメーター毎に差分をとる。
- 変化の有るパラメーターは1とdeltaの値を書き込む。
- 変化の無いパラメーターは0を書き込む。
- つまり変化しない部分は1ビットに圧縮できる。

UDPでどうやってデルタをとるのか？



過去数十フレーム分のステータスをリングバッファで残しておく
相手が既に受け取っている事が判明しているフレーム番号のステータスに
対してデルタをとればよい。

当たり判定など



FPSの場合、ユーザーが感じる同期ズレは当たり判定。

サーバーからのデーターが届くまでの遅延のせいで当たり判定がずれる。

どうしたら良いか？

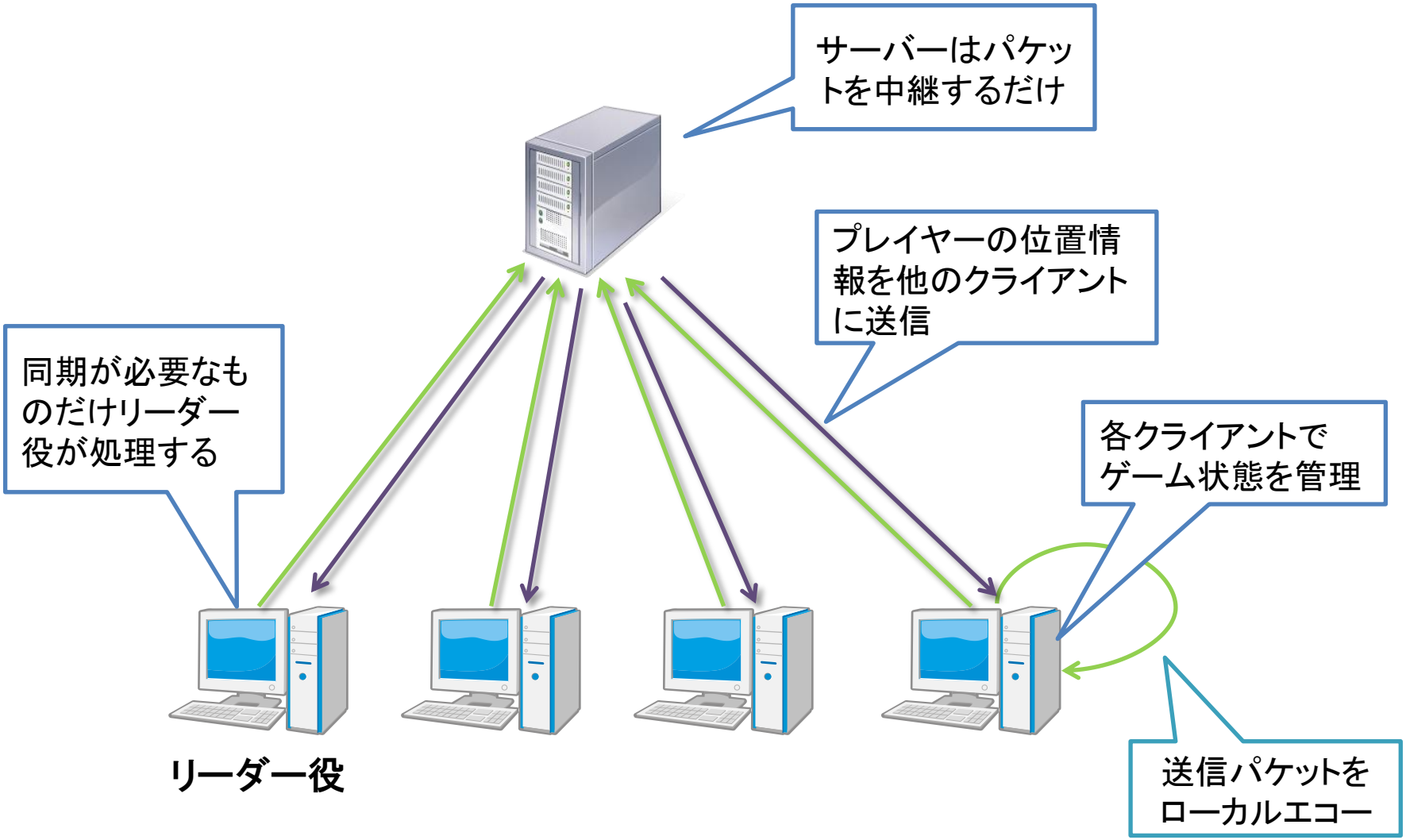
PSO、PSUの場合

Phantasy Star Online
2000年

PSOのゲーム要求

- 全世界対応で世界中の人と遊べること。
- ただし通信ラグを感じないようにしたい。
- アクション性やや高め。
- プレイヤー同士の対戦はなし。協力プレイのみ。
- サーバーは用意するがなるべく安く。

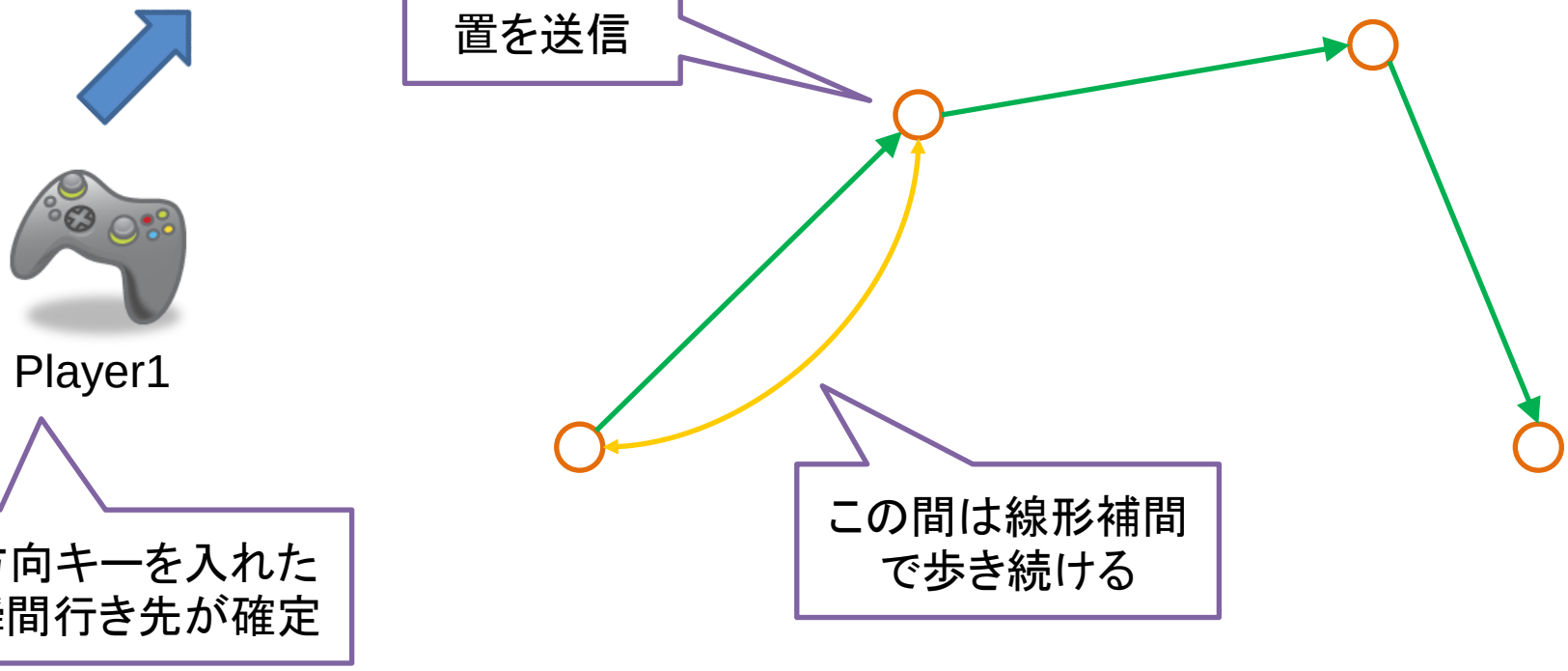
クライアント分散処理型



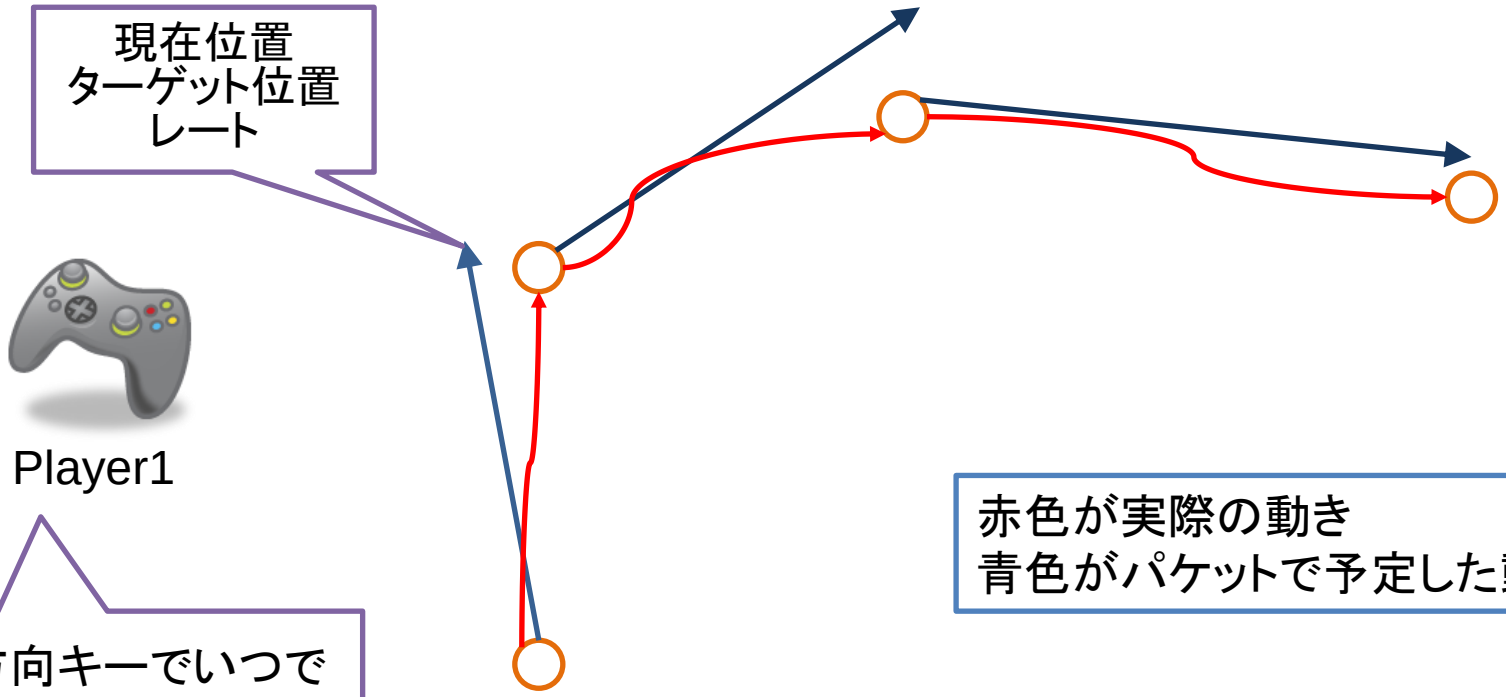
イベント発生型パケット

- 定期的にパケットを送信するのではなく、何か変化が起きた時だけパケットを送信。
- プレイヤーのアクション毎にパケットを送信するようなプログラムの書き方になる。
- 特に送受信のタイミング合わせはしていない。
- パケットを受け取ったら即実行。
- データーの送信量を減らすのに効果的。

PSOの移動パケット



PSUの移動パケット


現在位置
ターゲット位置
レート
Player1
方向キーでいつでも自由に動ける
赤色が実際の動き
青色がパケットで予定した動き

こんなの無理

エネミーの挙動

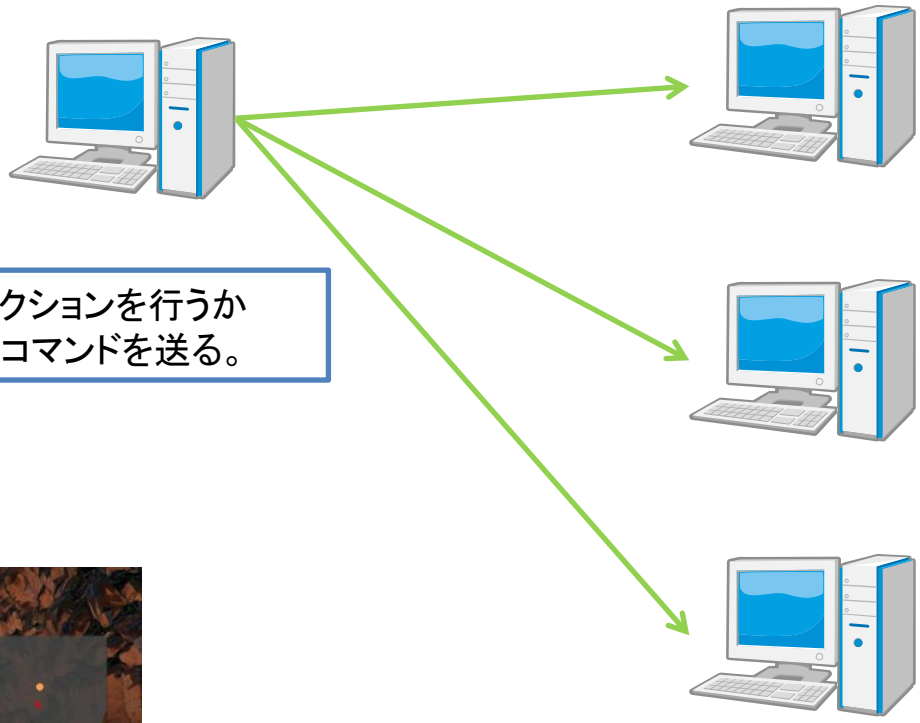
- 非同期型のゲームではAIや物理エンジン等と相性が悪い。
  - 入力が同期していないので結果が合わない。

- エネミー座標の同期を取らない。
  - ダメージなどの結果だけ送信

ボスAIの同期

歩行
回転
ミニジャンプ
ジャンプ踏み付け
ダッシュ
火球発射
火炎放射移動
突進
溶岩の挙動

1アクション10～30秒程度

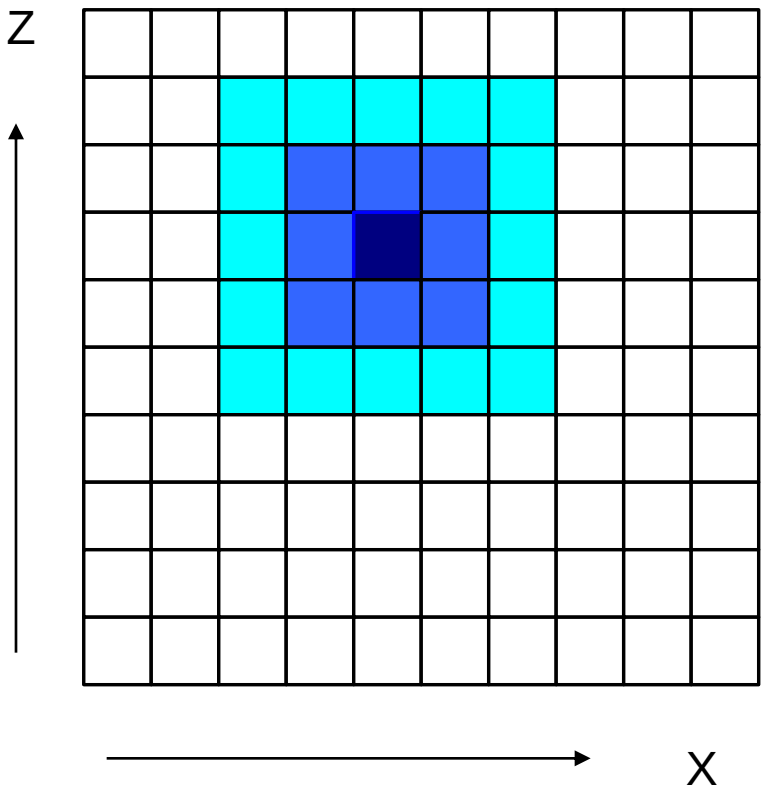


基本的に各アクションは同じ挙動になるようにする。
パラメーターは誰をターゲットにするか、
どこへ攻撃するかなど。
位置がずれたらジャンプして補正等を行う。

これでほぼ問題無い程度に同期する。

MMOエリアのしくみ

グリッドツリー



- MMOの為にパケット削減処理が必要
- X,Z軸座標10m単位のグリッドツリーを作成
- プレイヤーからの移動パケットが来る度にグリッドを移動する
- 自分のいるグリッド、その周りのグリッド、さらにもう一回りのグリッドでパケットの送信頻度を低下させる。1/1、1/4、1/16、といった感じ。

まとめ

- キー入力同期
  - 通信部分さえ出来れば簡単
  - 通信品質に左右される
  - 多人数対応は難しい
- コマンド入力同期
  - 間接的操作のゲームデザインが有効
  - ネットワーク負荷をAIでカバー
- サーバー集中処理
  - サーバーで処理するので同期が取りやすい
  - 通信を圧縮する仕組みが不可欠
- クライアント分散処理
  - 応答性能が高い
  - 同期に関してゲームデザインでカバー
  - プログラムは大変

ネットワークの今後

連絡先

ご静聴ありがとうございました。

講義に関する質問などはこちらまで。

Mail: Setsumasa_Akio@sega.co.jp

Twitter: akio_se