知能情報工学実験演習III
並列プログラミング
2017
サーバプログラムの要求仕様

Ver. 2017092900

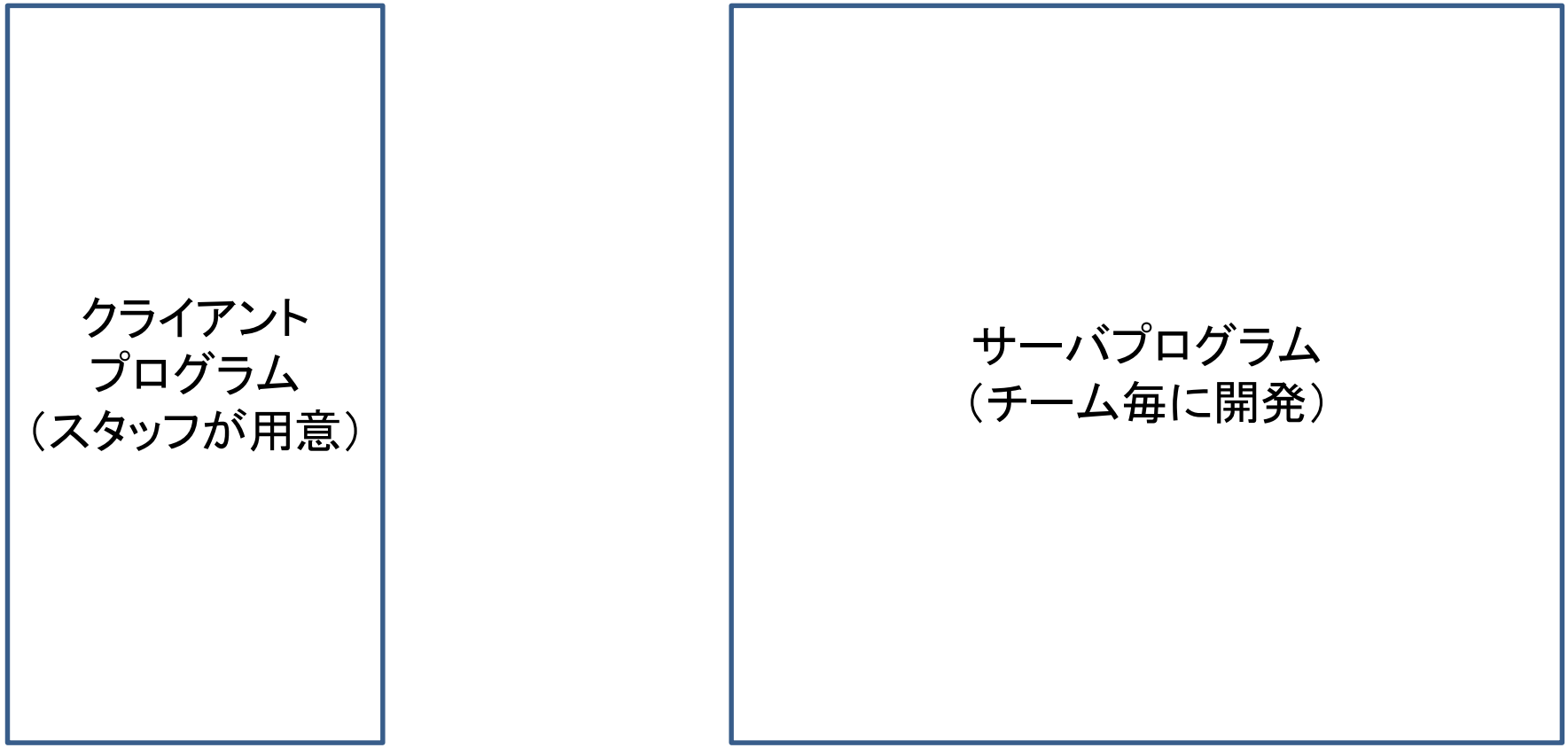
概要

- 次々に与えられる線形方程式を高速に解くサーバプログラムを作成してもらいたい
- 高速化のため、技巧を凝らして欲しい
  - メモリ階層の有効利用するための局所化
  - 複数のコアの同時利用（スレッド並列化）
    - 端末室の端末は2コア4スレッド、計算サーバは24/48
  - 複数データを一度に処理するSIMD命令の利用
  - 複数計算機の同時利用（分散並列化）
    - とりあえず、チーム人数程度を使う

解くべき線形方程式

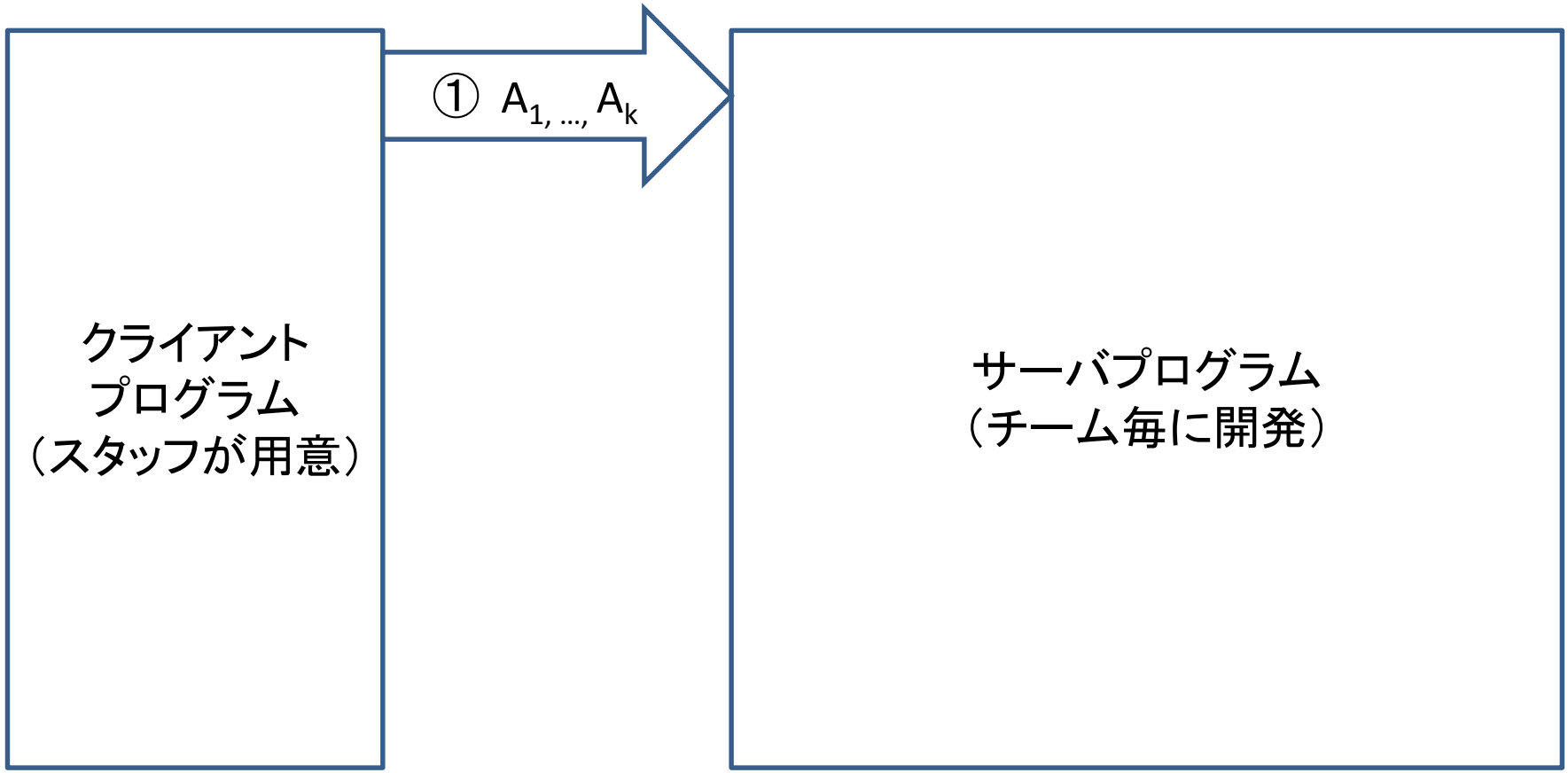
- ひとつの問題は
  - 行列 A と複数のベクトル $b_1, \ldots, b_l$ が与えられる
  - A は k 個の行列の積である: $A = A_1 \cdots A_k$
  - 各 b_i に対し、$A x_i = b_i$ を満たすベクトル x_i を求める
  - 解 x_i に許容される誤差は次の通り
    - $| (A x_i - b_i)/b_i |_\infty < 1e\text{-}8$
      - ただし、上記の割り算は要素ごとに行うもの
      - つまり、要素毎の相対誤差の最大値が 1e-8 未満であること
- この形の問題を大量に解くことが目的である

おおまかな流れ



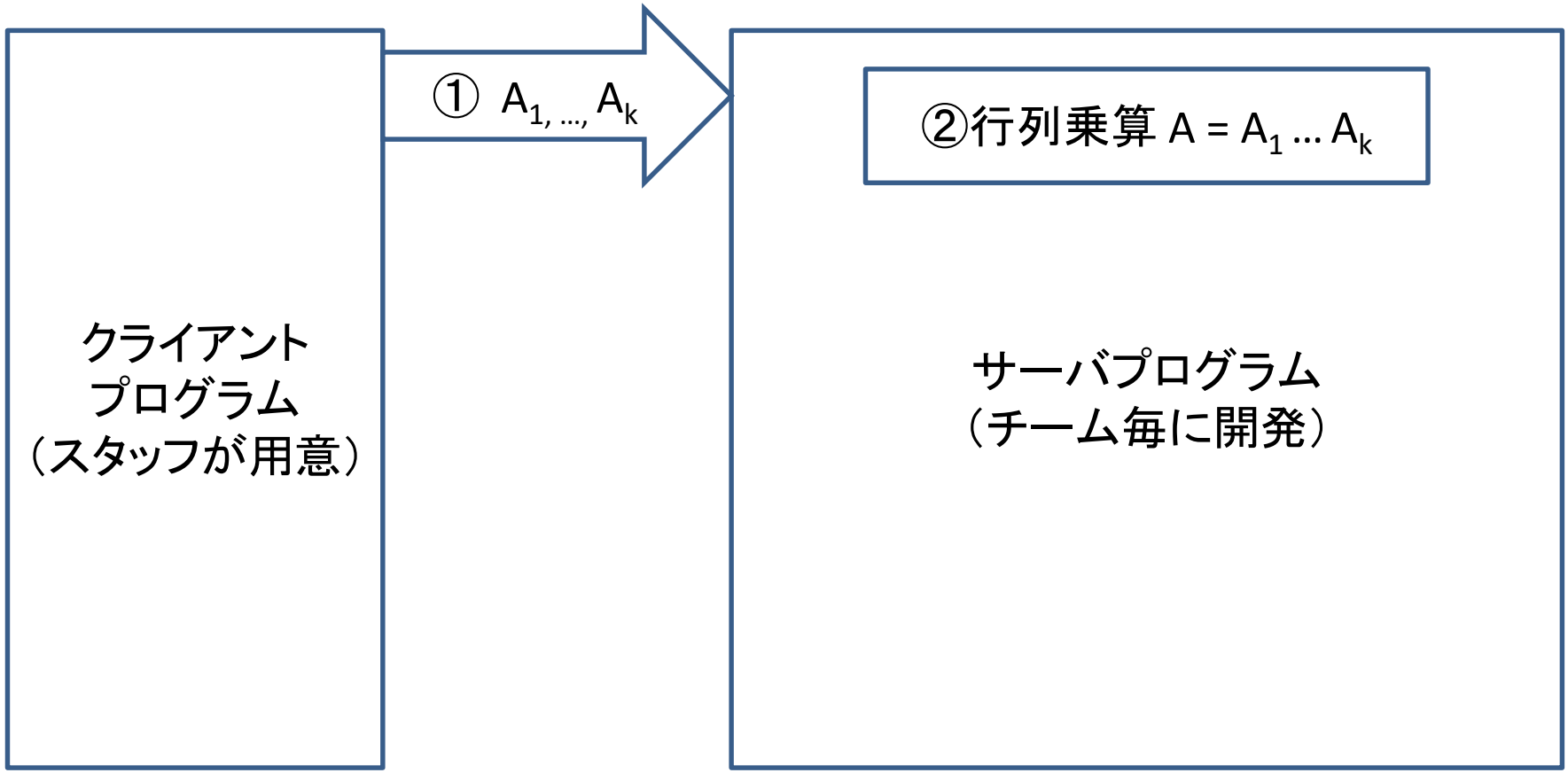
おおまかな流れ

① クライアントからサーバへ行列 $A_1, \ldots, A_k$ が伝えられる



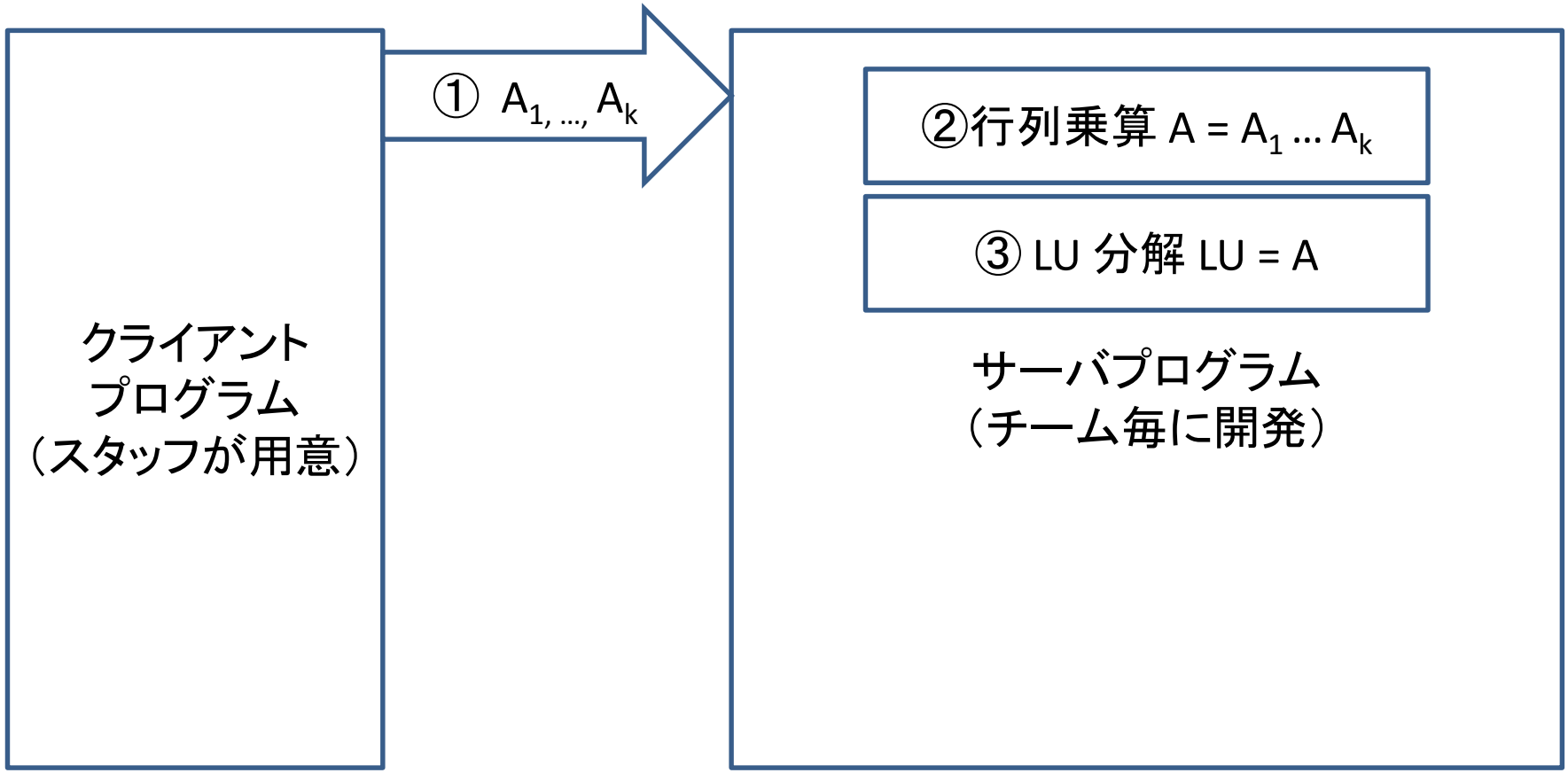
おおまかな流れ

② サーバが行列乗算を行い $A = A_1 \dots A_k$ を得る$O(n^3k)$



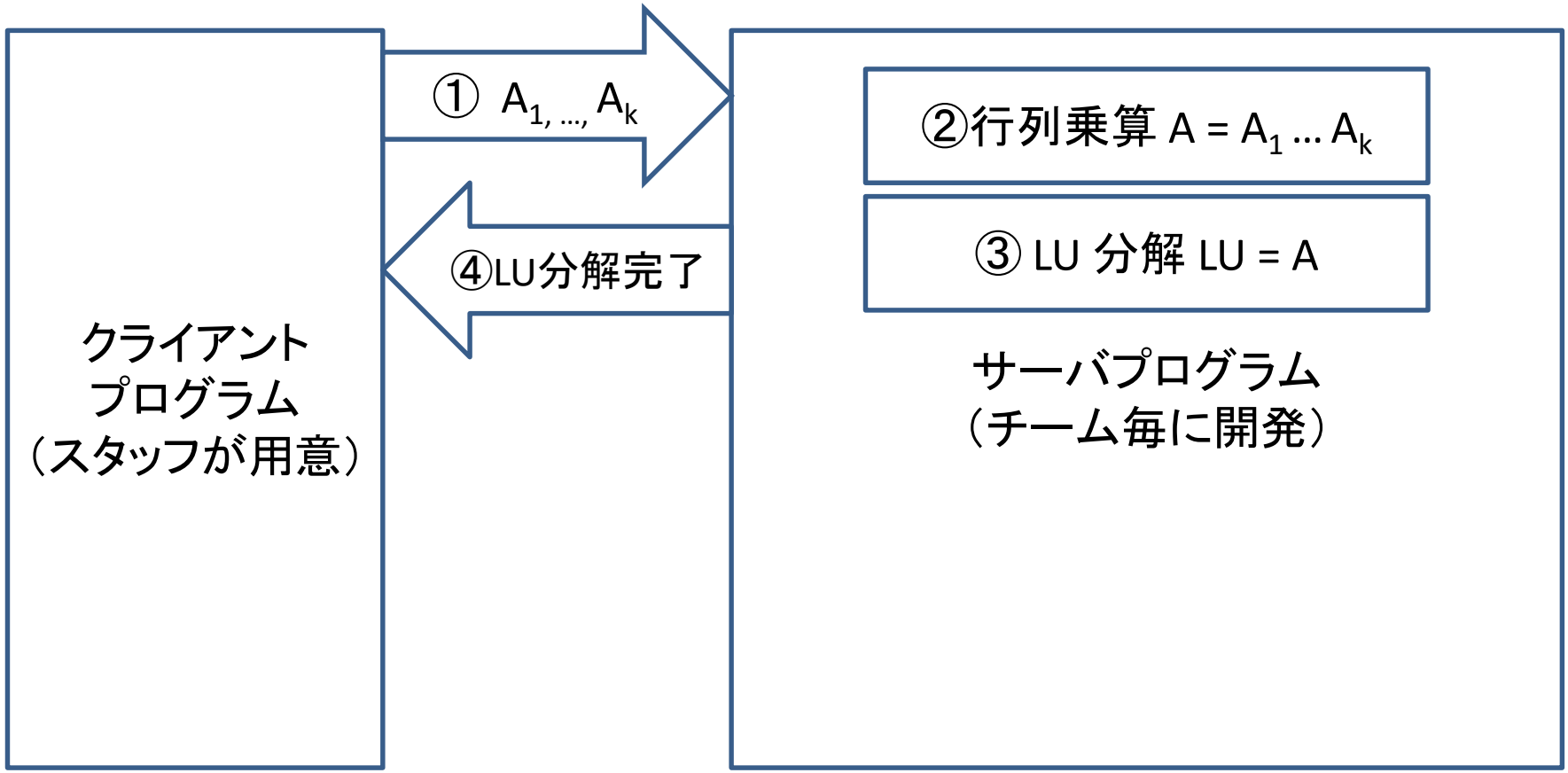
おおまかな流れ

③ サーバが A を LU 分解する$O(n^3)$



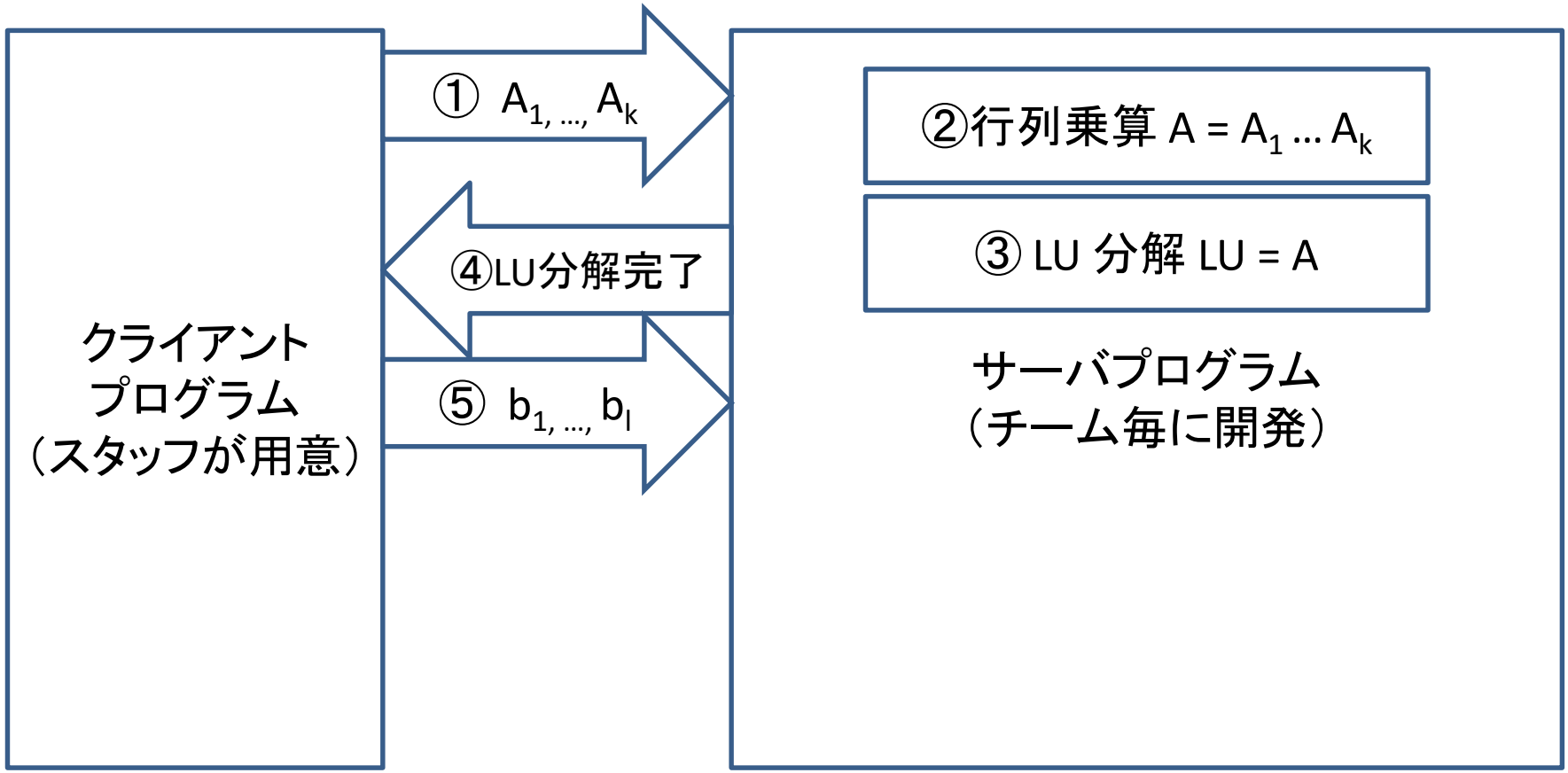
おおまかな流れ

④ サーバがクライアントに LU 分解完了を伝える



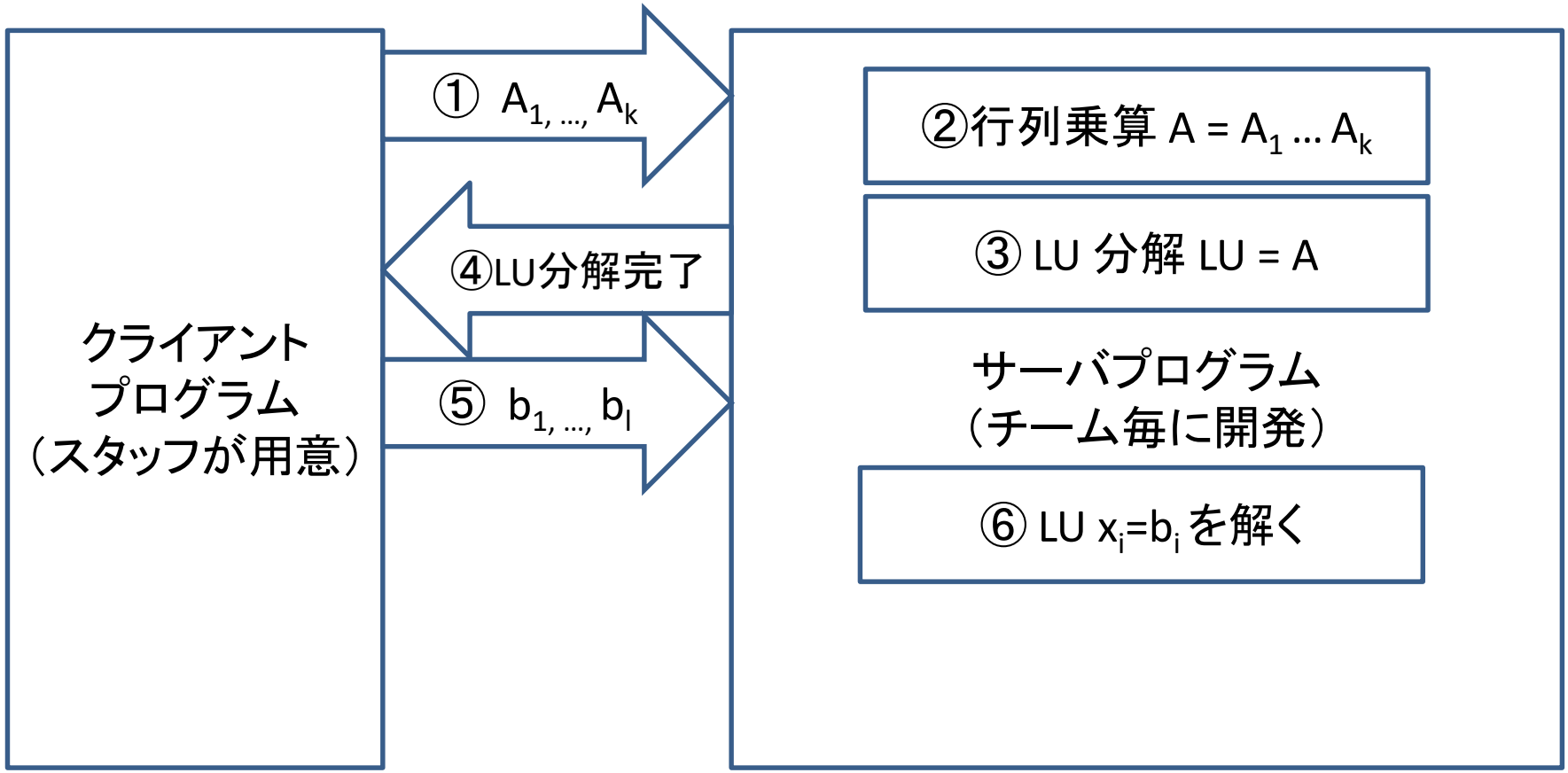
おおまかな流れ

⑤ クライアントがベクトル $b_1, \ldots, b_l$ を通知



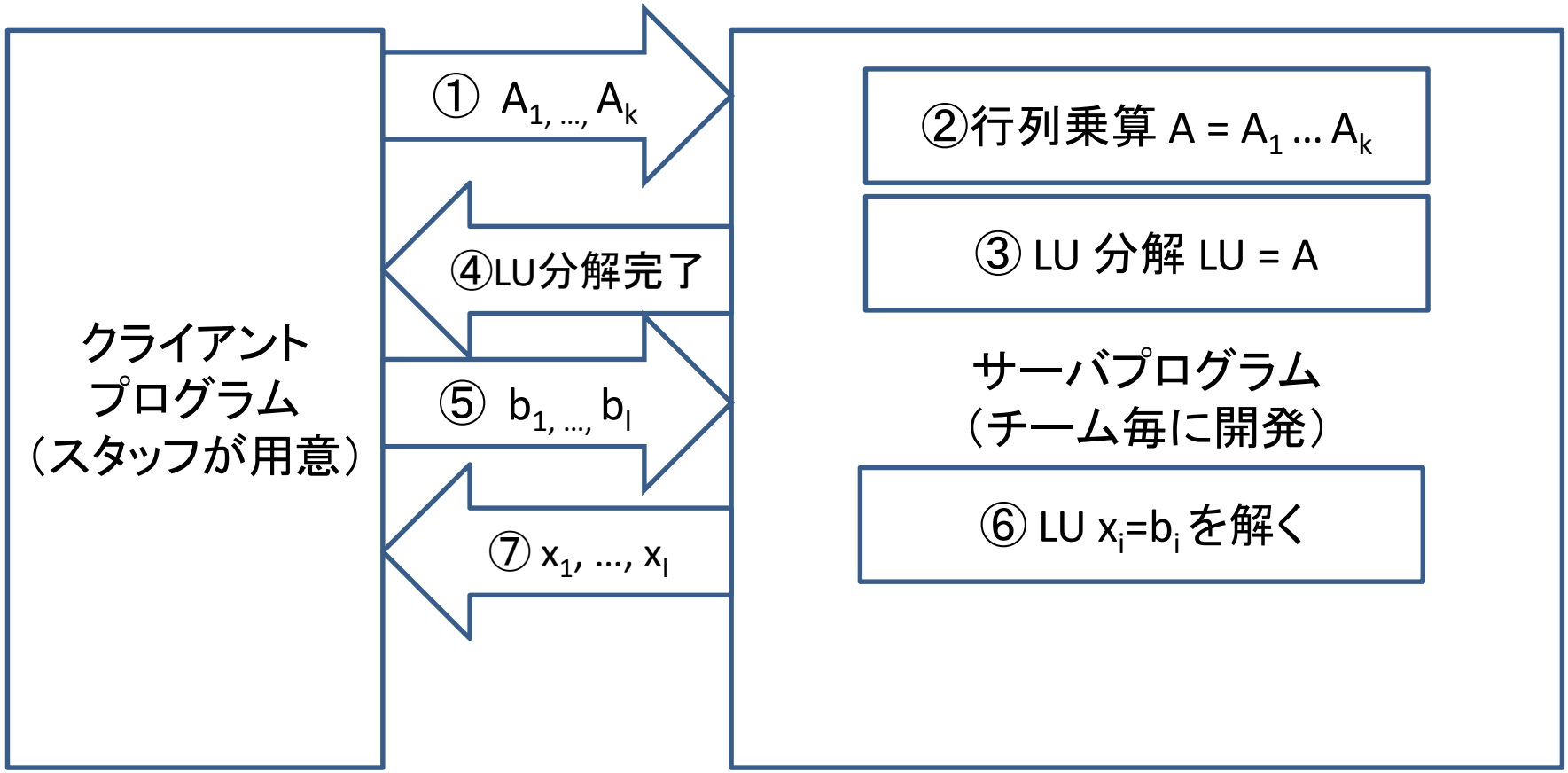
おおまかな流れ

⑥ LU分解結果を使って、解$x_1, \ldots, x_l$ を即座に求める$O(n^2 l)$



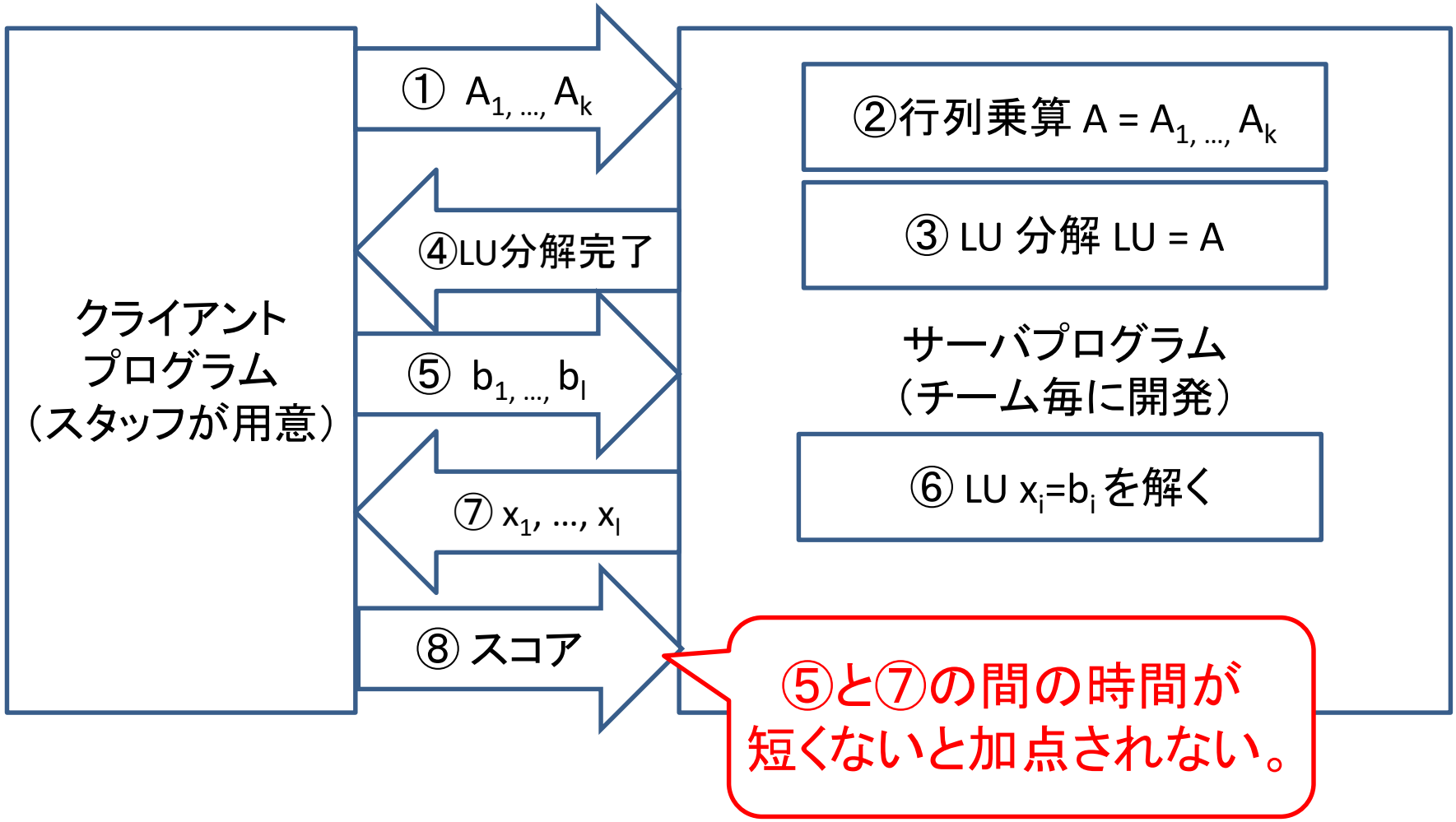
おおまかな流れ

⑦ サーバがクライアントに解 $x_1, \ldots, x_l$ を通知



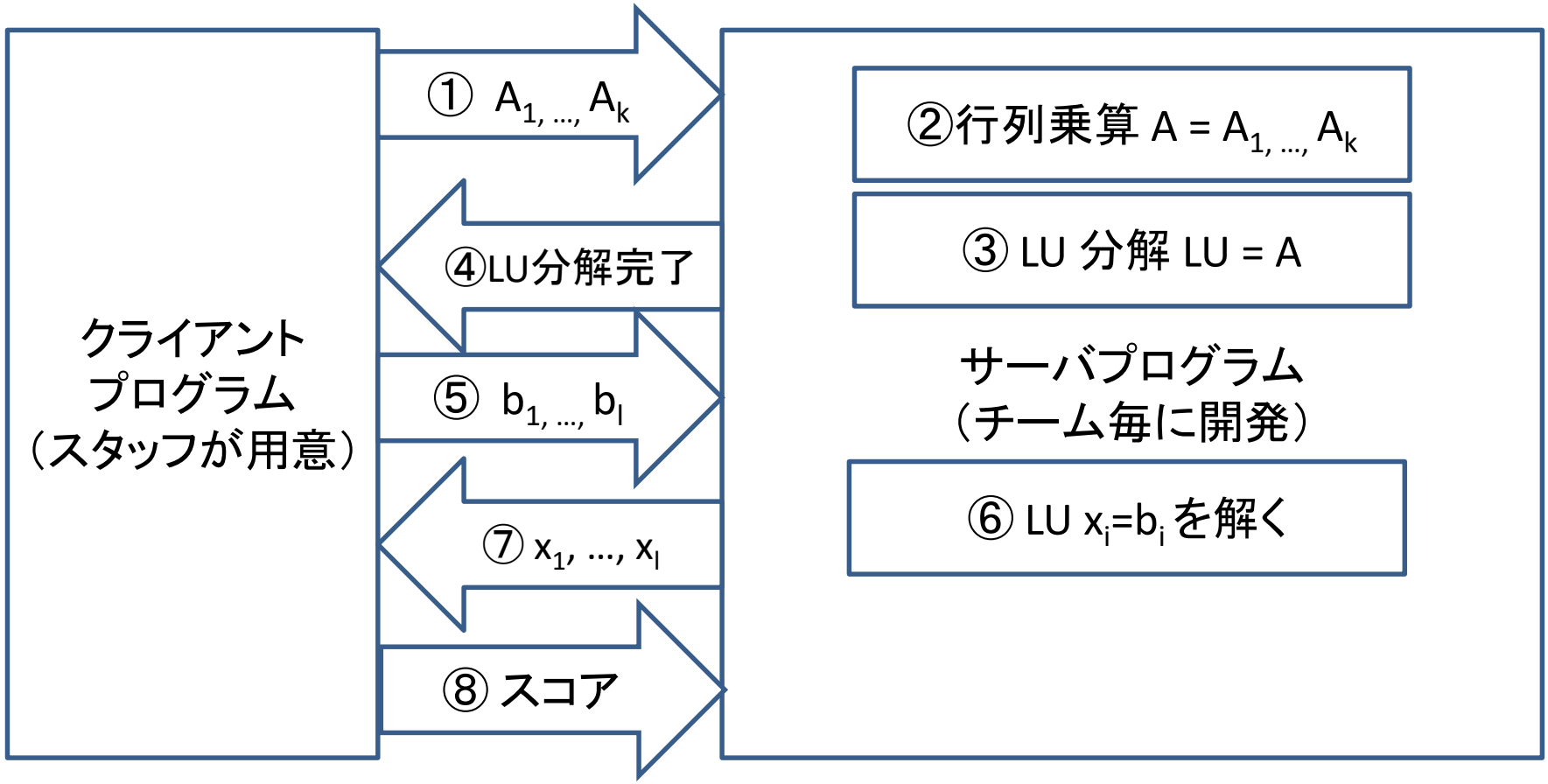
おおまかな流れ

⑧ クライアントがスコアを返す（競技用ですので）



おおまかな流れ

この流れを制限時間内で繰り返す。



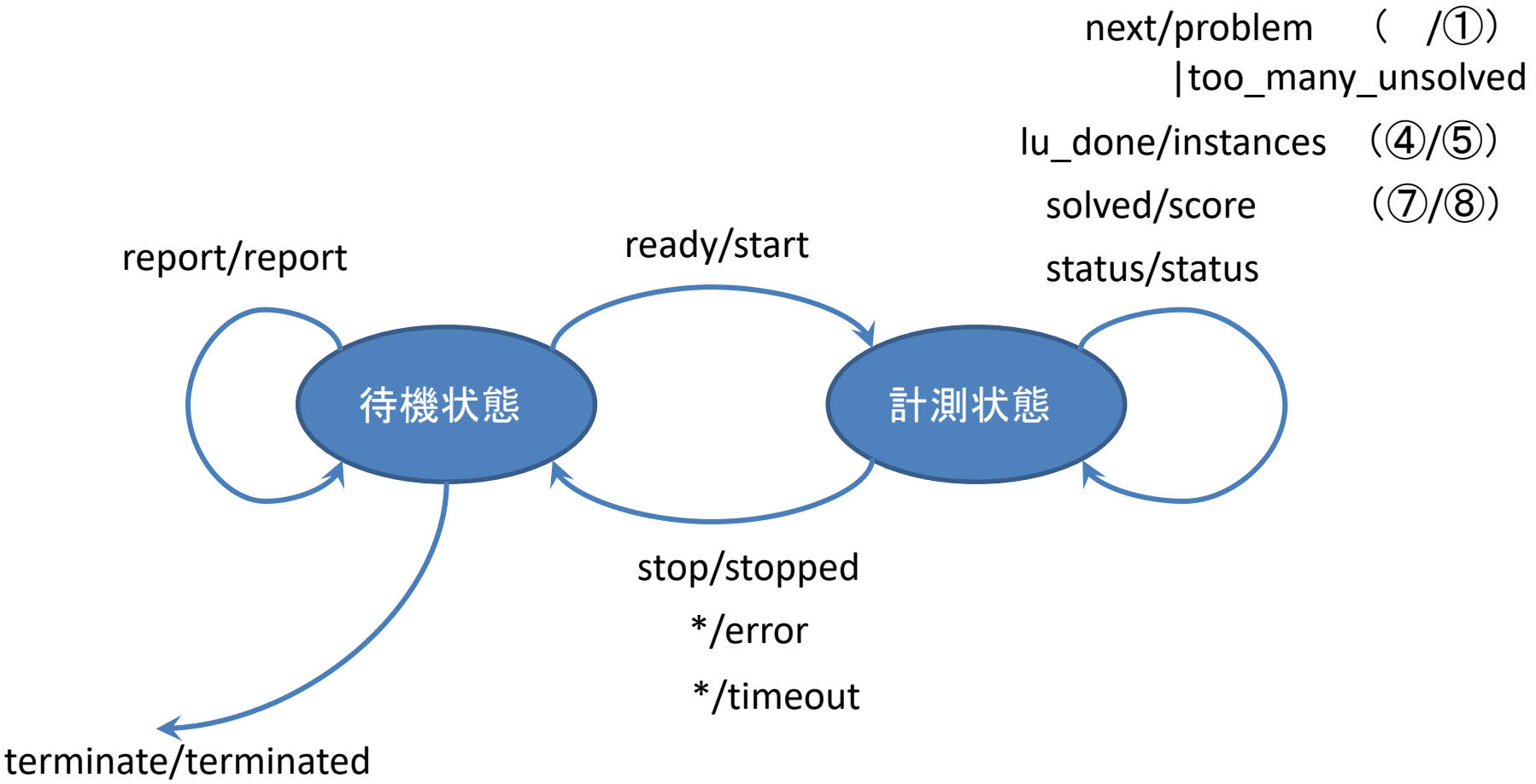
実際の注意点

- 行列 $A_1, .., A_k$ 及びベクトル$b_1,...,b_l$ の伝達
  - 実体を送らず、サイズと生成用パラメータで伝達
  - 生成ルーチンは後述
- ④や⑦の前に次の①を初めて良い
  - 使える計算機の台数分だけ先に問題を得ておくなどの戦略
  - 但し、未回答問題数に上限あり
- クライアントとサーバは別計算機でも良い

サーバクライアント間のプロトコル

- サーバクライアント間のやり取りは、「サーバからコマンド → クライアントからリプライ」という形となる
  - 通信と計算のサーバ・クライアント関係が逆
- 全てのトークンは空白で区切られている
- 全てのコマンドもリプライも一行
- クライアントは2状態をもつ
  - 待機状態
  - 測定状態

クライアントの状態遷移(command/reply)



ready/start

- S → C : “ready”
- C → S : “start”

- 計測開始
- クライアントは“start”の返信とともに計測状態へ移行
- クライアントのタイマーが動き出す

next/problem|too_many_unsolved（　/①）

- S → C : “next”
- C → S : “problem” *id n k* p_1 … p_k　#未回答数が少ない時
  - *id*, *n*, *k*, p_1 , …, p_k は10進正整数

  C → S : “too_many_unsolved”　#未回答数が多すぎる時

- サーバが新しい問題を要求
- クライアントが新しい問題の行列を伝達
  - 問題番号*id,* 各行列 A_i のサイズ *n* と生成パラメータ p_i
    - A_i = genmat(*n*, p_i)
    - 行列生成関数 genmat は後述
- 未回答問題が多い場合には問題は伝達されない

lu_done/instances （④/⑤）

- S → C : “lu_done” *id*
- C → S : “instances” *id* *l* p_1 … p_l
  - *id*, *l*, p_1 , …, p_k は10進正整数

- サーバが問題番号 *id* のLU分解終了を伝達
- クライアントが問題 *id* のベクトルを伝達
  - 各ベクトル b_i の生成パラメータ p_i
    - b_i = genvec(*n*, p_i)
    - ベクトル生成関数 genvec は後述
    - サイズ *n* は内部で記憶しておくこと

solved/score （⑦/⑧）

- S → C : “solved” *id* *l* $n_1\ x_{1,1} \ldots x_{1,n1}$ …… $n_l\ x_{l,1} \ldots x_{l,nl}$
- C → S : “score” *s*
  - *id*, *l*, n_i, *s* は10進正整数
  - $x_{i,j}$ は10進小数

- サーバが問題番号 *id* の答え $x_1, \ldots, x_n$ を伝達
  - ベクトル x_i 毎にサイズ n_i と要素 $x_{i,1} \ldots x_{i,ni}$ を列挙
- クライアントが現在のスコアを伝達
  - instances リプライからの経過時間が短い → 加点
  - スコアの詳細は後述

status/status

- S → C : “status”
- C → S : “status” 何らかの状況を示す文字列

- サーバが現在の状況を問い合わせる
- クライアントが現在の状況を返す
  - 文字列には空白や改行を含まない
  - 適宜エスケープを解除すること
  - エスケープは後述

stop/stopped

- S → C : “stop”
- C → S : “stopped”

- サーバが測定停止を要求
- クライアントが測定を停止し待機状態へ移行

report/report

- S → C : “report”
- C → S : “report” 前測定の結果を示す文字列

- サーバが直近の測定の結果を問い合わせる
- クライアントが直近の測定の結果を返す
  - 文字列には空白や改行を含まない
  - 適宜エスケープを解除すること
  - エスケープは後述

terminate/terminated

- S → C : “terminate”
- C → S : “terminated”

- サーバがクライアントに停止を要求
- クライアントはリプライ後に停止

*/timeout

- S → C : 測定状態で有効なコマンド
- C → S : “timeout”

- サーバが何らかのコマンドを発行
- クライアントは時間切れを通達し待機状態へ移行
  - つまり、クライアントは時間切れ後の最初のコマンドで待機状態へ移行する

*/error

- S → C : なんらかのコマンド
- C → S : “error” 何らかの情報の文字列

- サーバが何らかのコマンドを発行
- クライアントはエラーを通達し待機状態へ移行
  - エラー発生時、エラーを通知したらクライアントは待機状態へ戻る
  - 文字列には空白や改行を含まない
  - (適宜エスケープを解除すること)

行列生成関数 genmat

- C 言語のソースを示す。適宜各言語に翻訳せよ。

```
void
genmat(int n, mat_t a, int k)
{
  int i,j;
  double d, aij;
  int nk = n/k;
  double nk1 = 1./nk;
  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
      d = ((n - abs(i-j))/(double)n) / (abs(i - j) + 1);
      aij = d * ( (nk/2 - (abs(i - j) % nk))*nk1 + 1);
      a[i][j] = (0.1 * aij);
    }
    a[i][i] += 1;
  }
}
```

行列生成関数 genvec

- C 言語のソースを示す。適宜各言語に翻訳せよ。

```
void
genvec(int n, vec_t x, int k)
{
  int i;
  double dd = 2 * 3.14159265358979323846264338327 / n * k;
  for(i=0;i<n;i++){
    x[i] = ((i % 19) + 1) * ( sin(i*dd) + 1.1);
  }
}
```

エスケープ

- 空白や改行を含まないようにするため、以下の変換を行っている
  - ‘%’ → “%%”
  - 空白 → “%s”
  - 改行(LF) → “%n”

スコア

- 行列乗算及びLU分解の計算量が$O(n^3)$
  → サイズ n の問題の加点 $\propto n^4$
  - 大きな問題を解けばお得

- ベクトル1つ当たり 200ms 以内で正答したら $(n/100)^4$ を加点

クライアントプログラム

- プロトコルに基づいて動くクライアントプログラムを配布
- java -jar Client.jar -h
  - ヘルプやバージョンが見られます
  - オプションにはサイズや制限時間の設定があります
- ネットワークモード（指定ポートで待つ）

  Java -jar Client.jar -- -p 12345 -s 20 -vv
  - 12345 ポートで待つ、制限時間20秒、詳細メッセージ
- 子プロセス標準入出力対話モード（子プロセスを起動）

  java -jar Client.jar -- -s 20 -e -vv -- ./lu
  - 子プロセス ./lu を起動、制限時間20秒、
    子のエラー出力をリダイレクト、詳細メッセージ

実装手法及び環境について

- 使用言語は何でもOK
  - SIMD命令使うには C/C++ 程度が必要
- 使用可能計算機
  - 各チームに割り当てた端末室の端末（～６台）
  - 計算サーバ1台
    - 詳細は後ほど
    - 他の端末との混合での利用は難しいかも
      → 端末用実装と計算サーバ特化の実装

想定問題サイズ等

- 1200 <= n <= 1600
- 1 <= k <= 10
- 1 <= l <= 10
- 制限時間 <= 60s
- 未回答問題数上限 = 10