安全な計算状態操作機構のこれまでと今後

八杉 昌宏

京都大学 情報学研究科

(基盤研究(B)「安全な計算状態操作機構の実用化」研究代表者)

発表内容

- (安全な)計算状態操作機構L-closureとは?
  - AT との関係
- これまでの応用例
- これまでの実装
- 最近の実装
- 最近の応用例
- これからの課題

L-closure: 計算状態操作機構とは? (1/2)

- 高水準言語のコンパイラの中間言語で提供される言語機構として提案
  - 高信頼，高性能な高水準言語の実装
- 実行中のソフトウェアの動的再構成・保全
  - 呼び出し元で眠っている変数への合法的アクセス
  - 汎用性が高く，さまざまに利用可能
  - ごみ集め，負荷分散など
  - 表面的ではなく抜本的に途中で見直せる
  - 自分自身を対象としたデバッガを操作するように

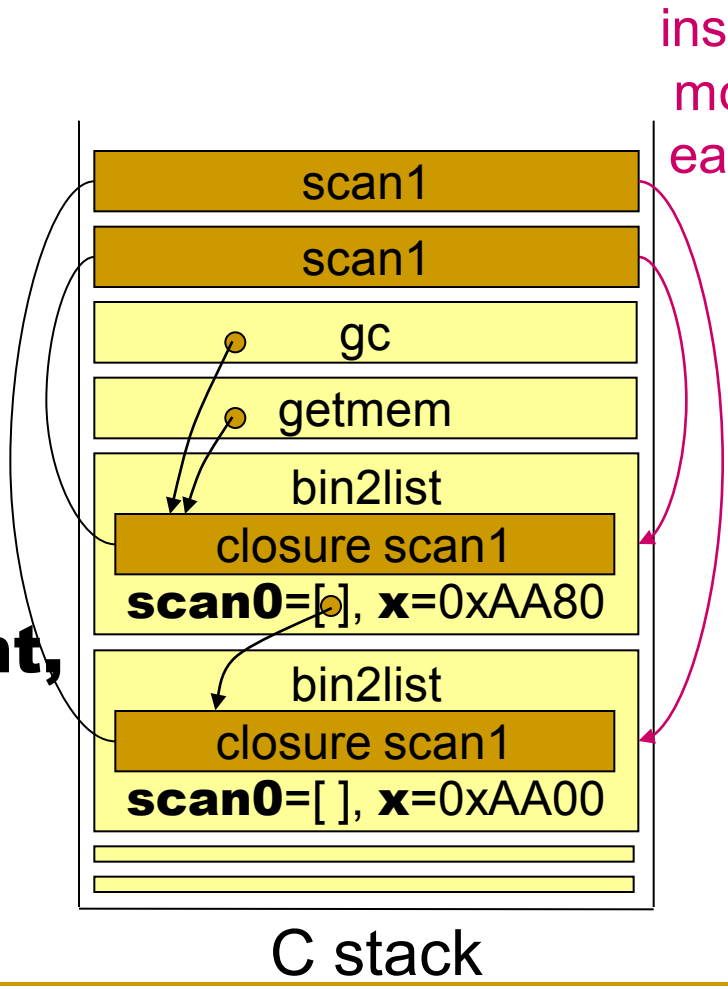
L-closure: 計算状態操作機構とは? (2/2)

- 拡張C言語の仕様として
- 入れ子関数(クロージャ)を利用
  - 呼出し元に眠る変数の値への安全で正式なアクセス
- 高度な技法で高性能実装
  - アクセス対象となる変数もレジスタ割り当て候補
  - 実際に呼び出すまでクロージャの初期化を遅延
  - 2つの実装
    - GCC拡張による実装 [八杉 他 CC2006, IPSJ PRO論文誌2008 (平成21年度論文賞)]
    - 標準C言語への翻訳方式 [平石・八杉他 IPSJ PRO論文誌2006]

```
Alist *bin2list(void (*scan0) closure (move_f),
                Bintree *x, Alist *rest){
  Alist *a = 0; KVpair *kv = 0;
 void scan1 closure (move_f mv){  /* create closure */
   x = mv(x); rest = mv(rest);    /* scan roots */
   a = mv(a); kv = mv(kv);        /* scan roots */
   scan0(mv);                     /* scan older roots */
  }
 // pass pointer to closure "scan1" on the following calls.
  if(x->right) rest = bin2list(scan1, x->right, rest);
  kv = getmem(scan1, &KVpair_d);    /* allocation */
  kv->key = x->key; kv->val = x->val;
  a = getmem(scan1, &Alist_d);      /* allocation */
  a->kv = kv; a->cdr = rest;
  rest = a;
  if(x->left) rest = bin2list(scan1, x->left, rest);
  return rest;
}
```

Using Lexical Closures

```
/* simplified for explanation */
bin2list(closure_t scan0,
     Bintree *x, ...){
     void scan1 closure (){
          x = move(x);
          scan0();
     }
     bin2list(scan1, x->right,
     ...);
     getmem(scan1,... );
}
```



実行時(抜本的)細部選択の自由 (1/2)

- 同じ計算内容で複数手段
  - 負荷分散: ある計算を逐次実行? 並列実行? いつ, どこで実行?
- 期待できる最良の選択は?
  - 局所的観点からは「逐次実行」を選択
  - それればかりだと, 並列実行の機会なし → 「合成の誤謬」的な事態に

実行時(抜本的)細部選択の自由 (2/2)

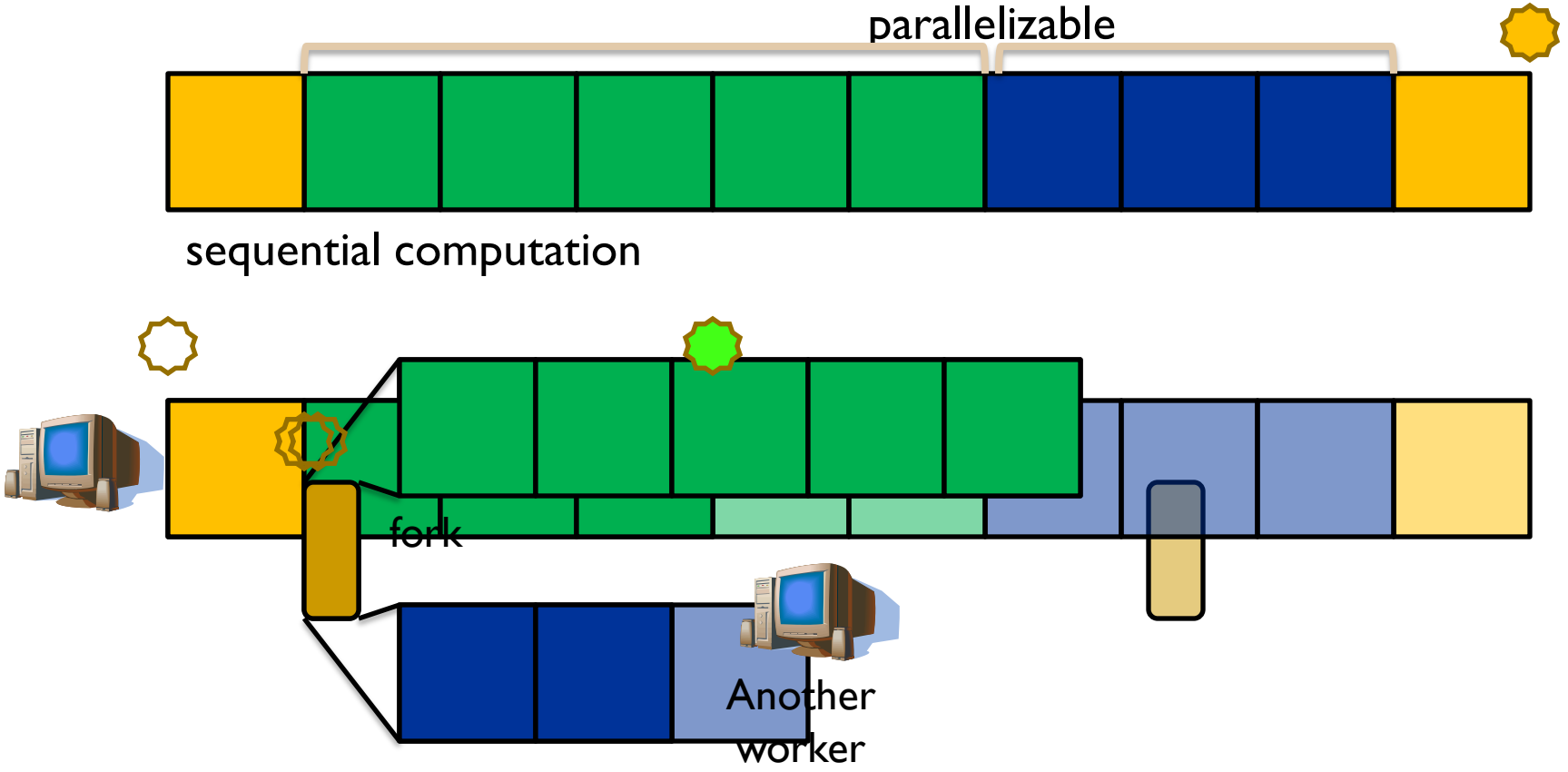
- ある種のジレンマ
  - あるrunで，常に同じ選択ではいけない
- ゲーム理論の混合戦略(ある確率で選択)は?
  - 選択すべき時にはまだ情報不足
- 後から選択を変更できるとよいが?
  - 計算状態操作機構で可能
    (呼び出し元の変数の値の参照・変更などにより)
- 負荷分散の例
  - 逐次実行を後から並列実行に変更

発表内容

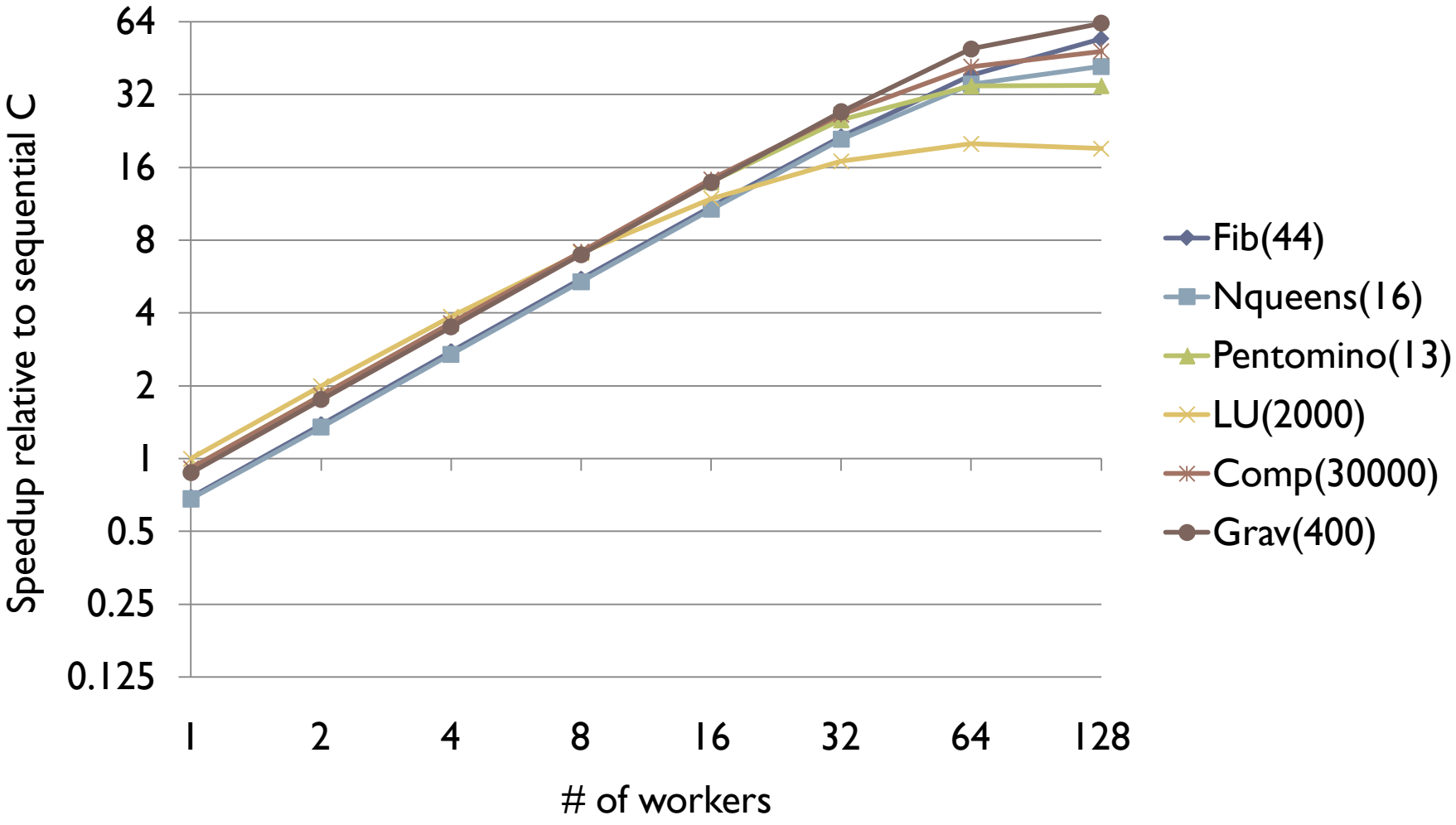
- (安全な)計算状態操作機構L-closureとは?
  - AT との関係
- これまでの応用例
  - Tascell: Backtracking-based Load Balancing [平石・八杉 他 PPoPP 2009]
- これまでの実装
- 最近の実装
- 最近の応用例
- これからの課題

Tascell: Load Balancing Framework [PPoPP2009]

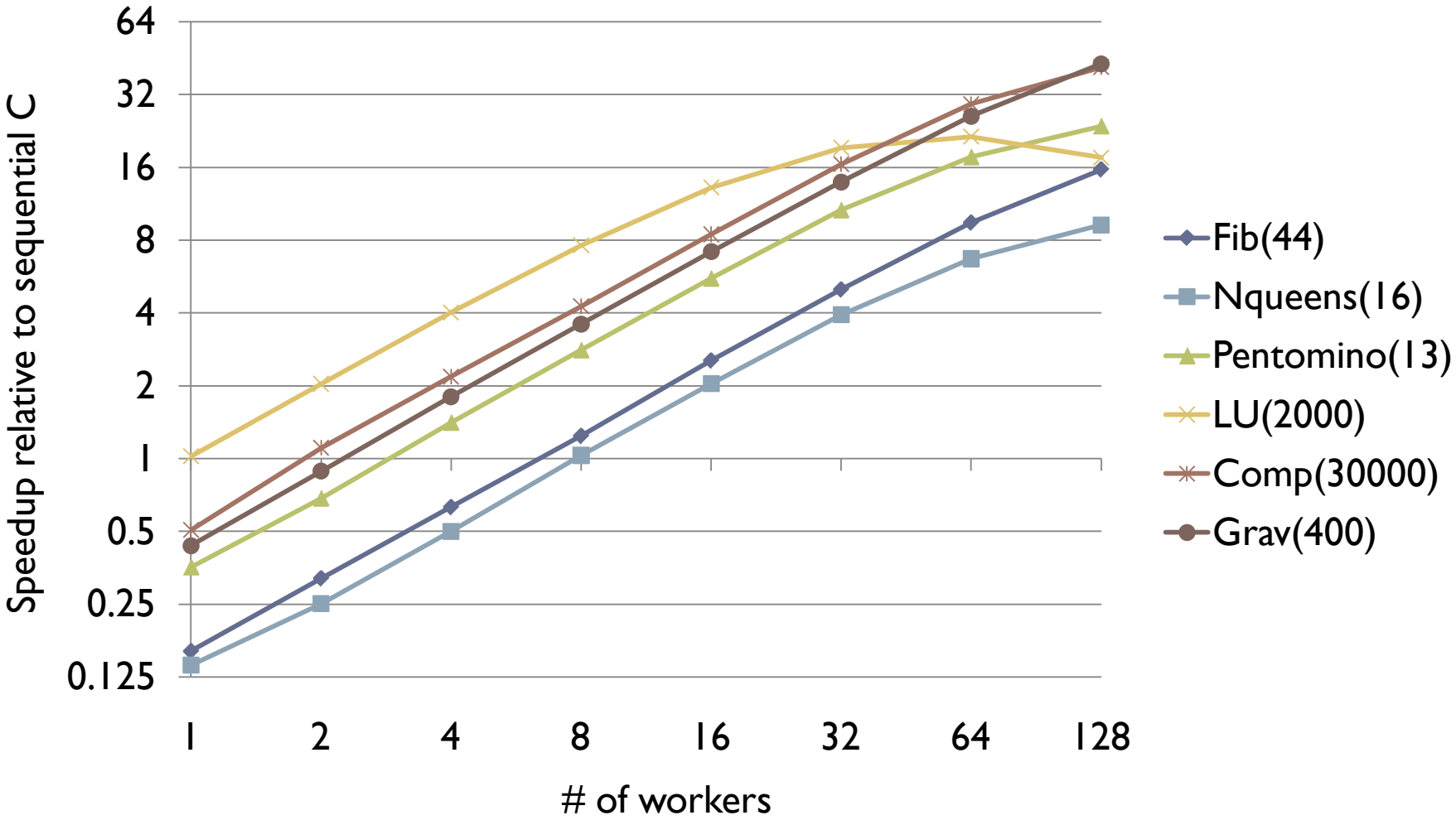
- *"Logical thread"-free*
- Compute sequentially, fork only when requested



測定結果（Niagara2 Tascell）



測定結果（Niagara2 Cilk [PLDI'98]）

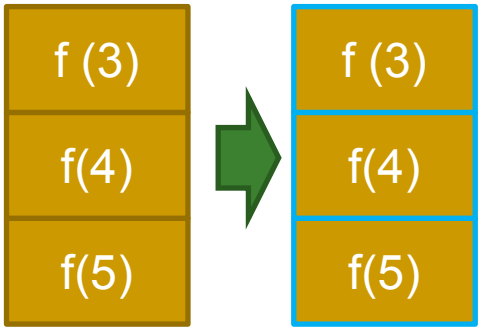


発表内容

- (安全な)計算状態操作機構L-closureとは?
  - AT との関係
- これまでの応用例
- これまでの実装
  - 標準C言語への翻訳
  - GCC拡張の実装モデル
    - GCC 3.4.6ベース, SPARC, IA-32
- 最近の実装
- 最近の応用例
- これからの課題

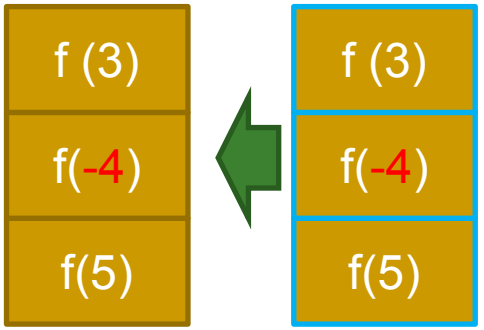
Transformation-based implementation of L-closures [Hiraishi,Yasugi,Yuasa2005,6]

- Translating into “standard” C
- Using an explicit stack
  - Save C frames as explicit frames and “return”
  - Call the L-closure which inspects/modifies data on the explicit frames



Transformation-based implementation of L-closures [Hiraishi,Yasugi,Yuasa2005,6]

- Translating into “standard” C
- Using an explicit stack
  - Save C frames as explicit frames and “return”
  - Call the L-closure which inspects/modifies data on the explicit frames
  - “Call” and restore C frames from the *modified* explicit frames



GCC-based Implementation Model for L-Closures

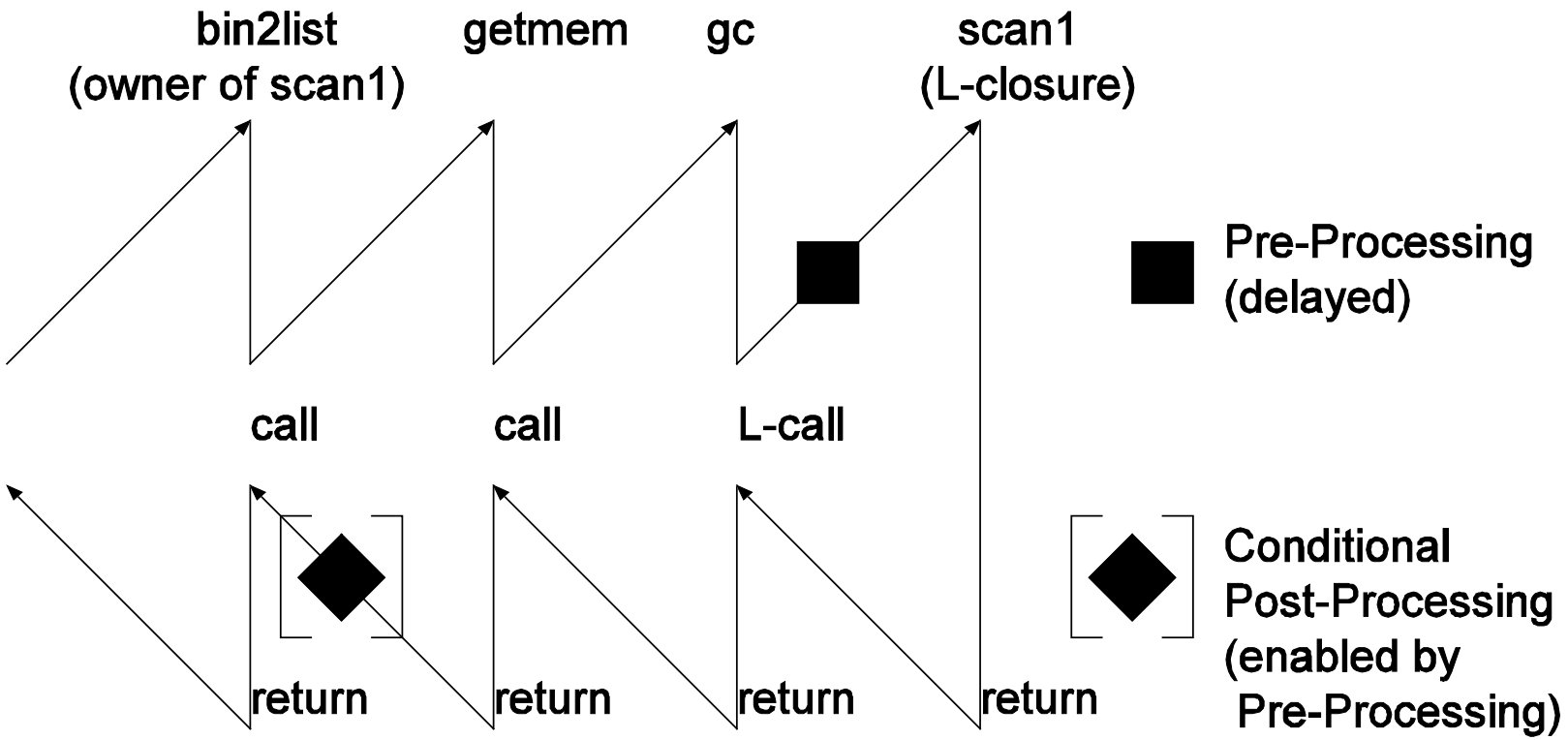
- Two locations for each variable
  - *Private* location
    - Compiler's pseudo register
    - Private values may be kept in registers.
  - *Shared* location
    - Stack frame slot
    - Accessed by L-closure bodies.
- The coherency between private and shared locations is maintained *lazily*.

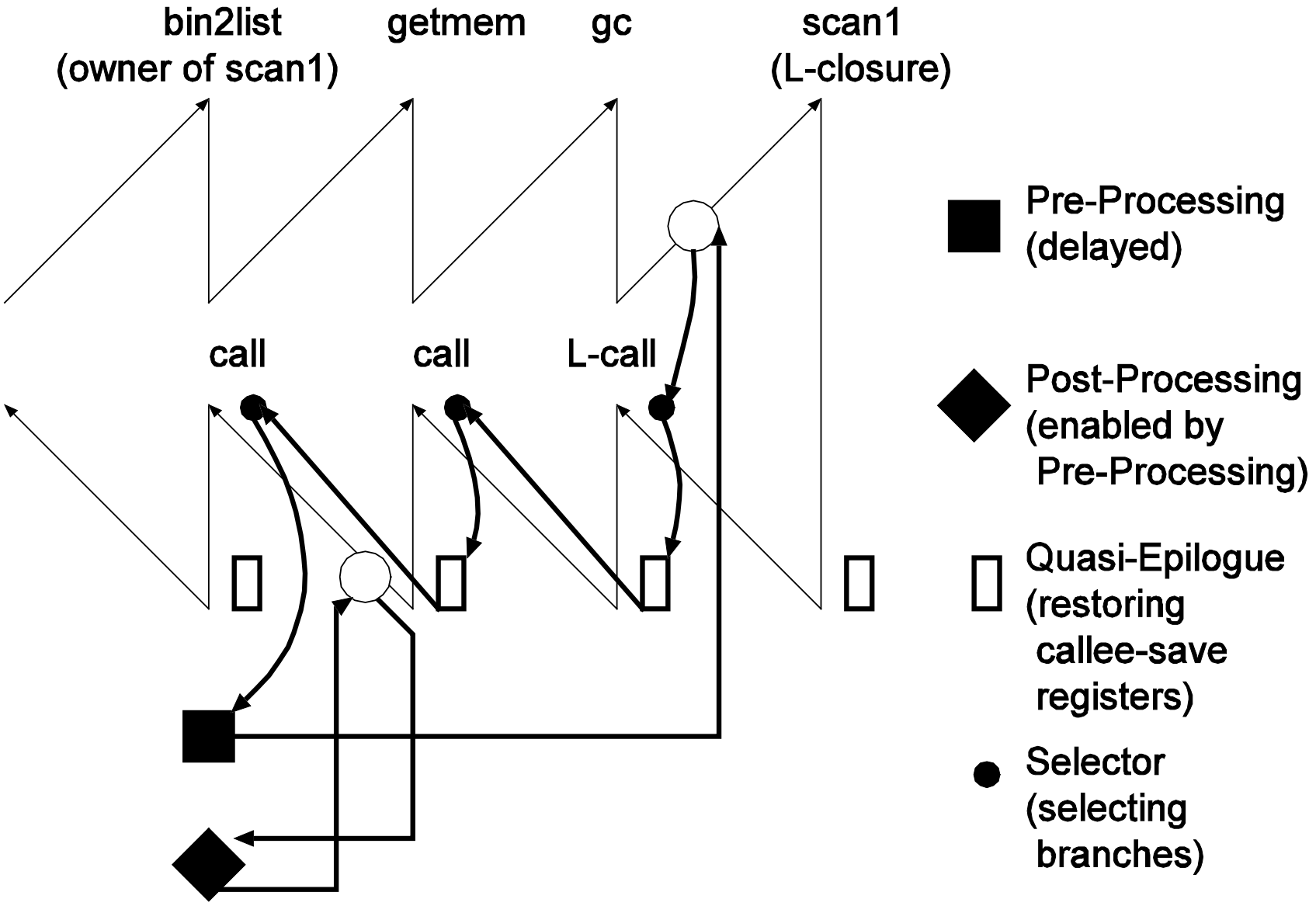
Implementation Model for L-Closures

- For efficiency, Pre-processing should be *delayed* until the L-closure is *actually invoked*.
- Delayed (lazy) pre-processing
  - Save private values ➔ shared locations
  - Initialize the code-environment pair
  - Enable post-processing
- Conditional post-processing
  - Restore private values ➔ shared locations

Lazy Pre-/Post-Processing for L-Closures

- Delayed until the L-closure is actually invoked




bin2list
(owner of scan1)
getmem
gc
scan1
(L-closure)
call
call
L-call
Pre-Processing
(delayed)
Post-Processing
(enabled by
Pre-Processing)
Quasi-Epilogue
(restoring
callee-save
registers)
Selector
(selecting
branches)

発表内容

- (安全な)計算状態操作機構L-closureとは?
  - ATとの関係
- これまでの応用例
- これまでの実装
  - 標準C言語への翻訳
  - GCC拡張の実装モデル
    - GCC 3.4.6, SPARC, IA-32
- 最近の実装
  - x86-64 のサポート，callee-save レジスタ利用増加は逆効果も
- 最近の応用例
- これからの課題

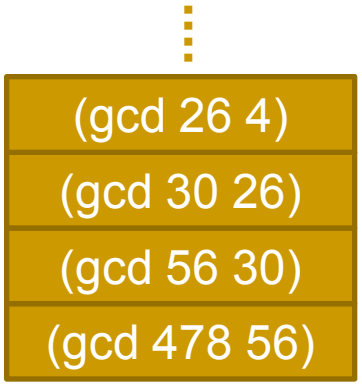
発表内容

- (安全な)計算状態操作機構L-closureとは?
  - AT との関係
- これまでの応用例
- これまでの実装
- 最近の実装
- 最近の応用例
  - Tascell: spanning tree construction [ICPADS 2010]
  - 真の末尾再帰 (proper tail recursion) [ILC 2010]
- これからの課題

関数型言語と再帰

- Scheme 言語等，副作用抜きで反復計算を表現
- 例: ユークリッドの互除法による最大公約数

```
(define (gcd a b)
    (if (= b 0) a
        (gcd b (remainder a b))))
```

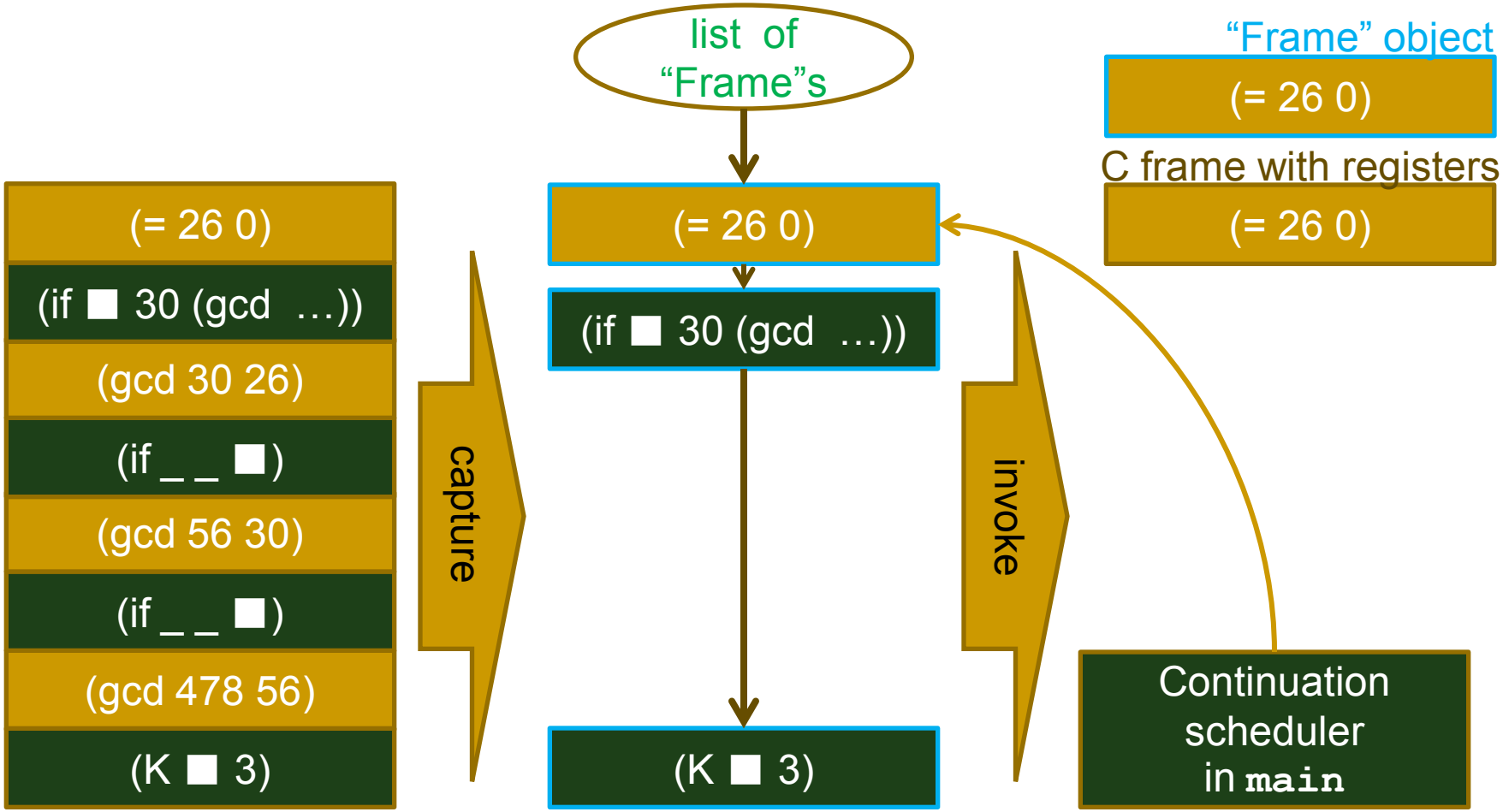


- 再帰呼び出し，末尾呼び出し
- 空間(space)を一定以上必要としないと期待
- 真の末尾再帰 (proper tail recursion)

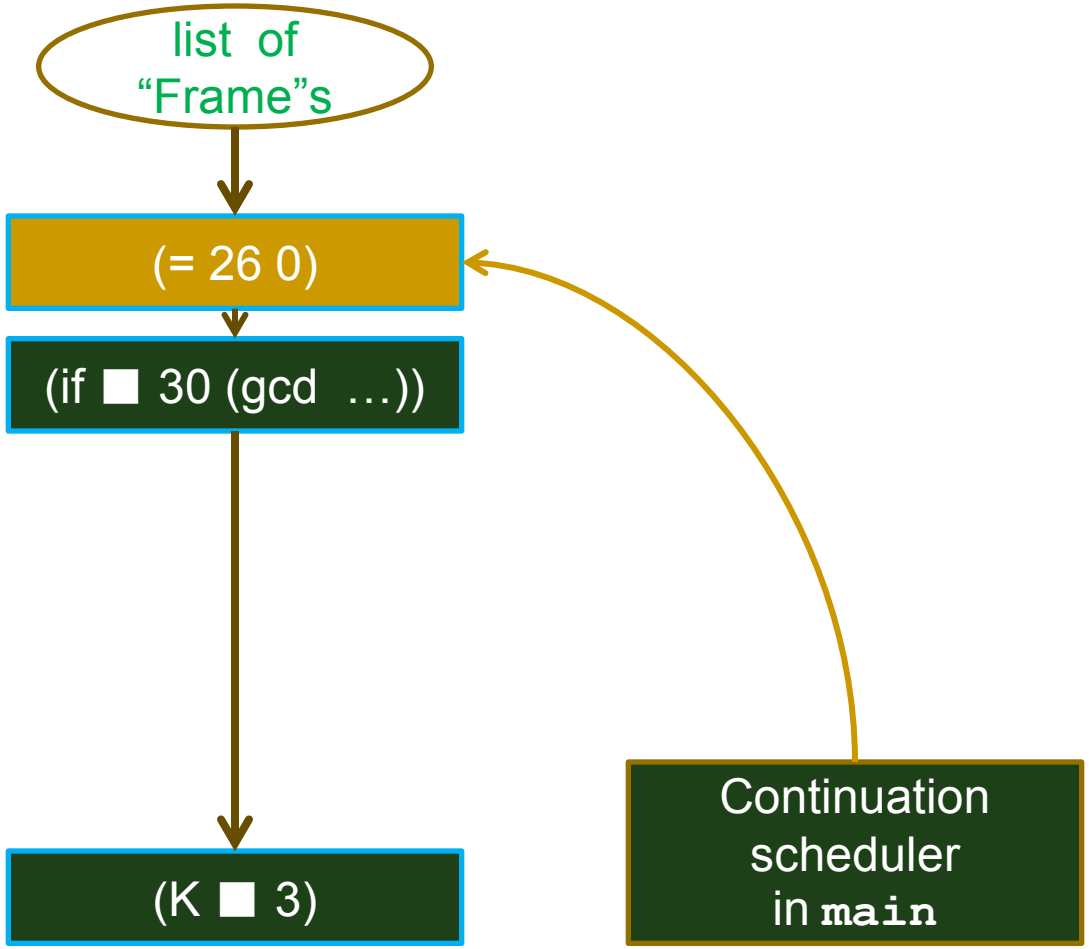
提案，基本アイディア，背景

- 提案: 拡張C言語による，真に末尾再帰的なSchemeインタプリタの実装手法
- Cの実行スタックが溢れそうになれば，残りの計算に必要な“Fame”オブジェクトのみを含むリストとして表現された空間効率の良い一級継続を生成し，即呼び出す (Clinger[PLDI’98]の定義を満たす)
- Schemeインタプリタ: JAKLD[湯淺’03]
  - Cで再実装[加藤’07]，トランポリン，継続は脱出のみ
- L-closure (を備えた拡張C言語XC-cube)
  - ごみ集め(GC)，一級継続，真の末尾再帰の実現

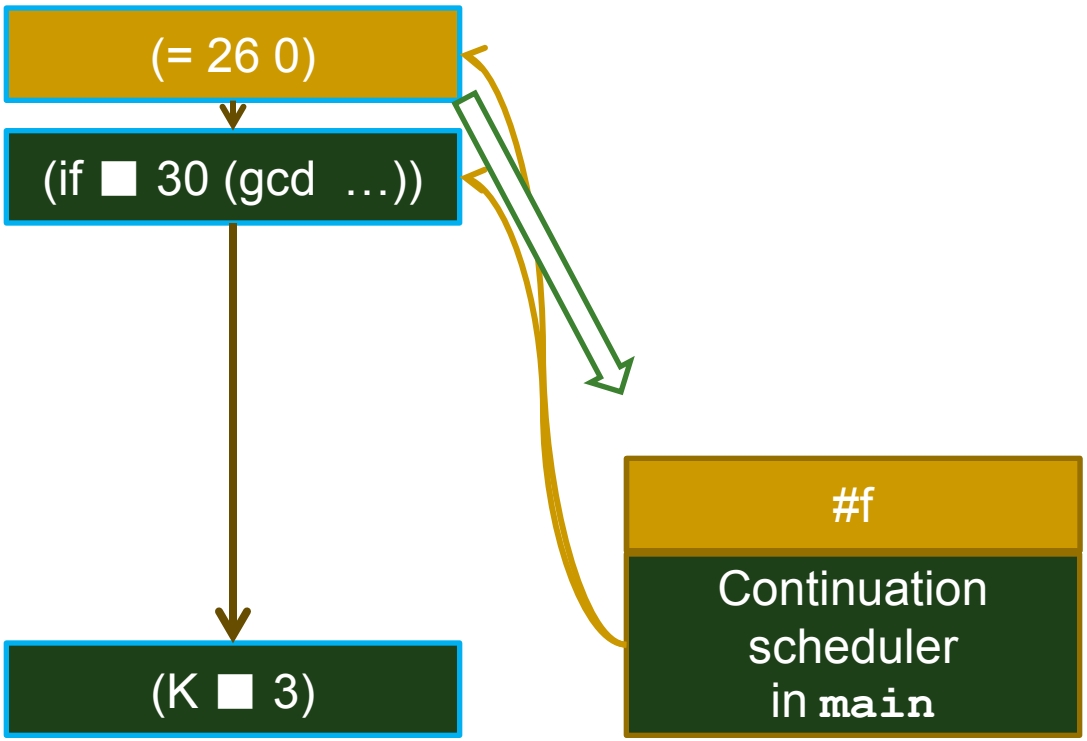
A space-efficient first-class continuation represented as a list of "Frame" objects



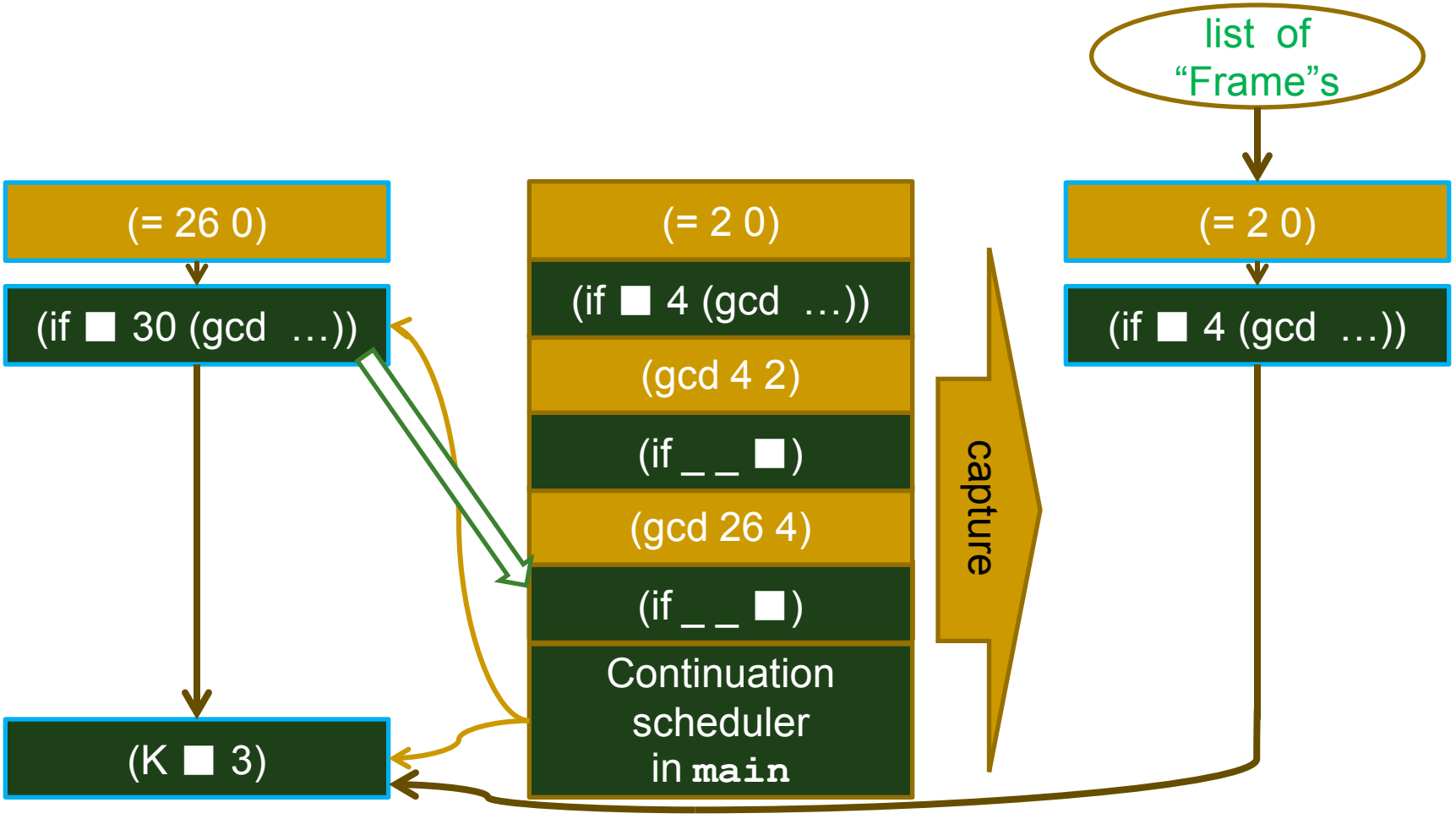
A space-efficient first-class continuation represented as a list of "Frame" objects



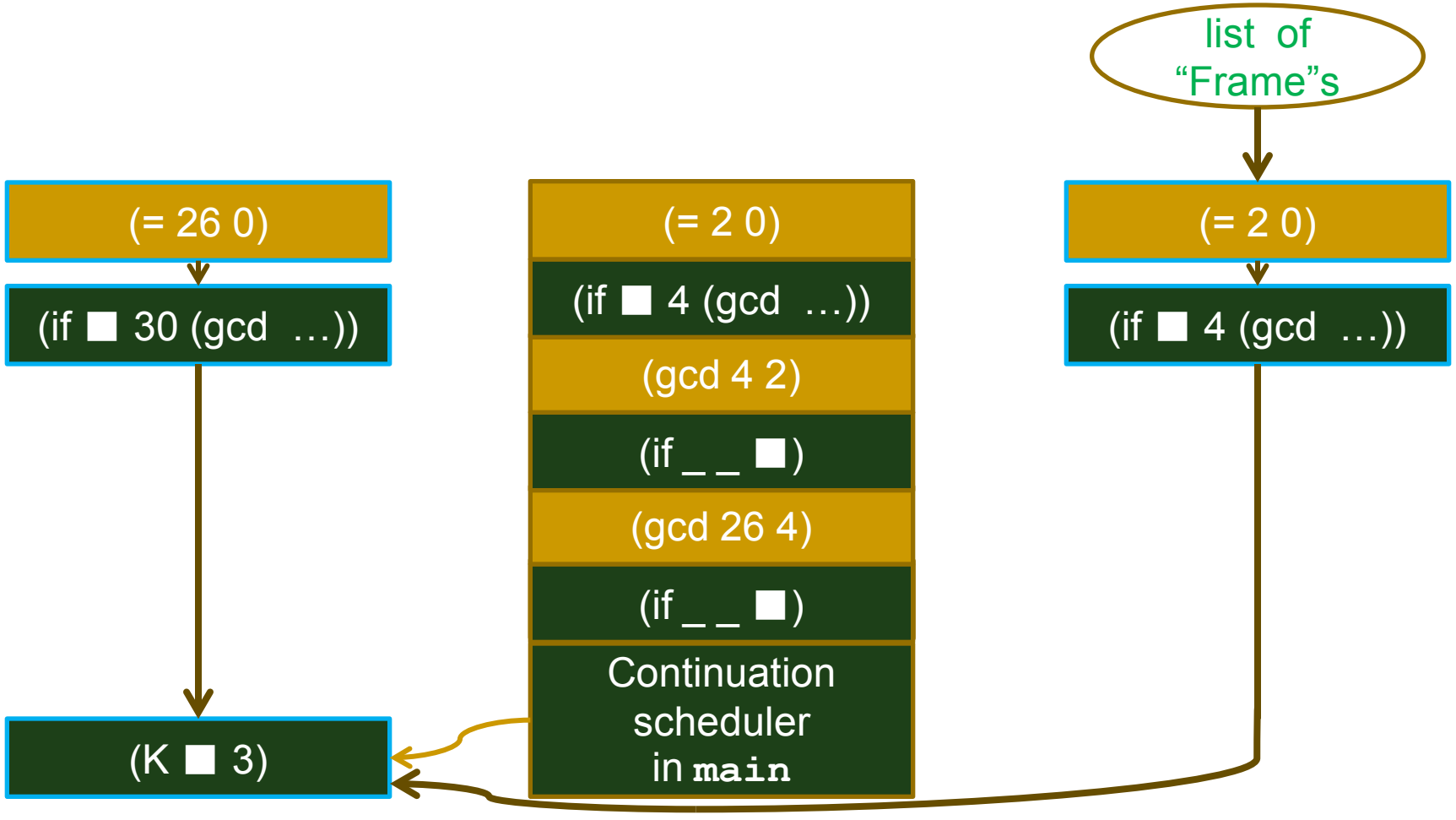
A space-efficient first-class continuation represented as a list of "Frame" objects



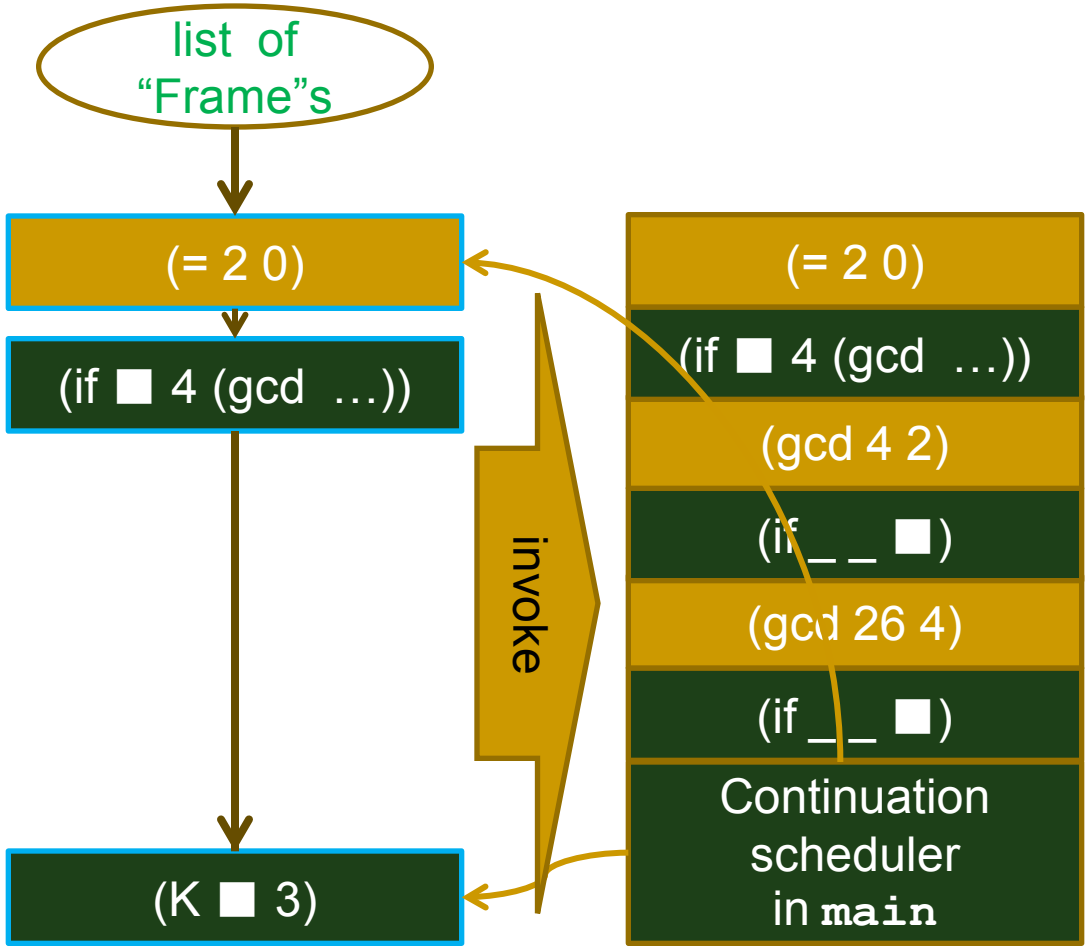
A space-efficient first-class continuation represented as a list of "Frame" objects



A space-efficient first-class continuation represented as a list of "Frame" objects



A space-efficient first-class continuation represented as a list of "Frame" objects

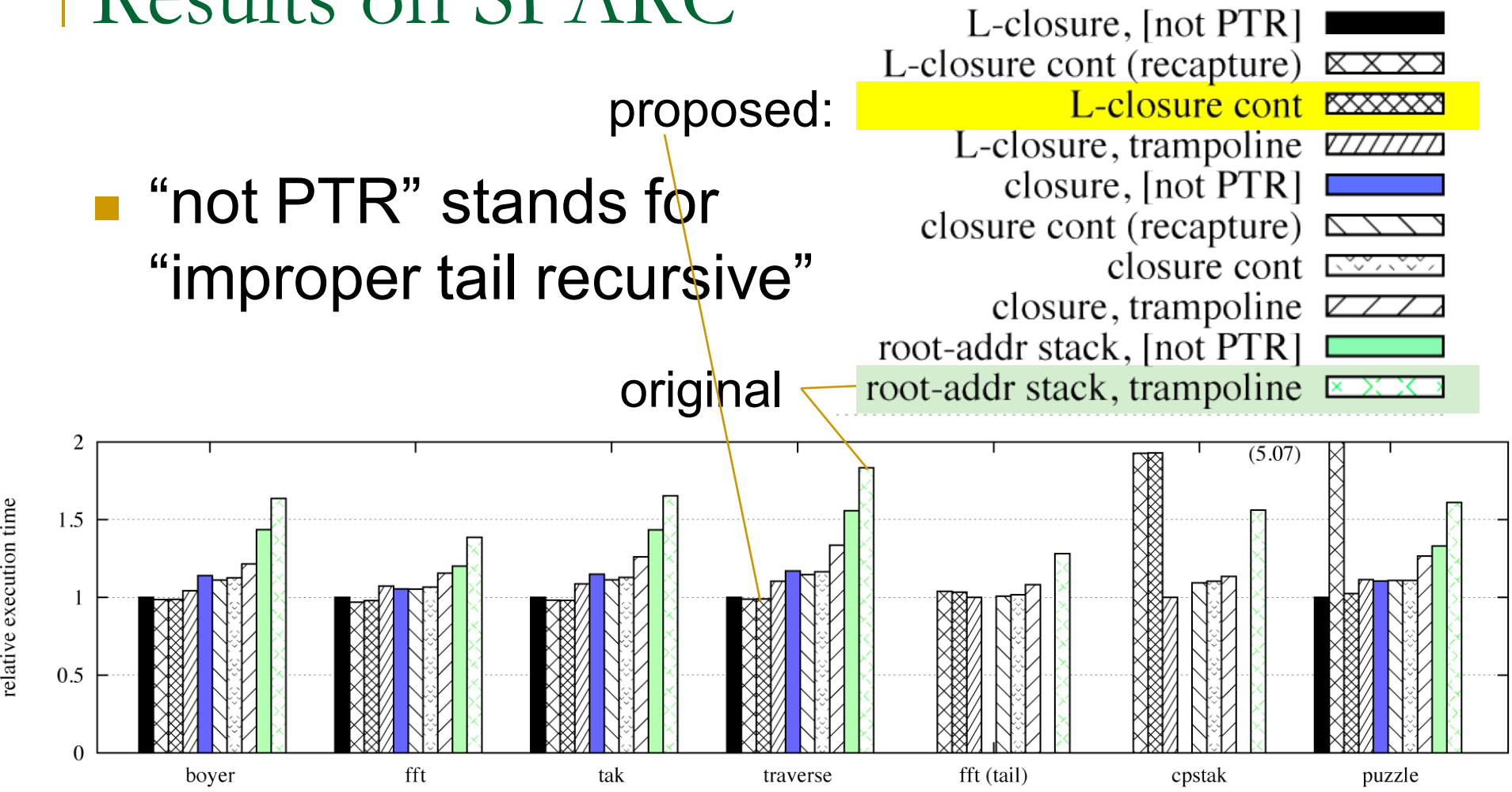


提案方式の基本アイディア (主要な貢献)

- 「空間効率の良い一級継続を生成し，即呼び出す」
  - これを何らかの基準で実行スタックが溢れそうになったら行い，実行スタックの利用をリセット
  - 実行スタックのための空間の大きさを一定（適当な定数）以下にとどめるという基準を設ける
  - 定数差以上には空間消費せず形式的定義を満たす
- 「とりあえずのものを定数サイズで利用」
  - キャッシュなども含む一般的アイディア
  - どのように利用を止めるかがポイント
  - Cの実行スタック中の計算に必要なデータを救出したい…

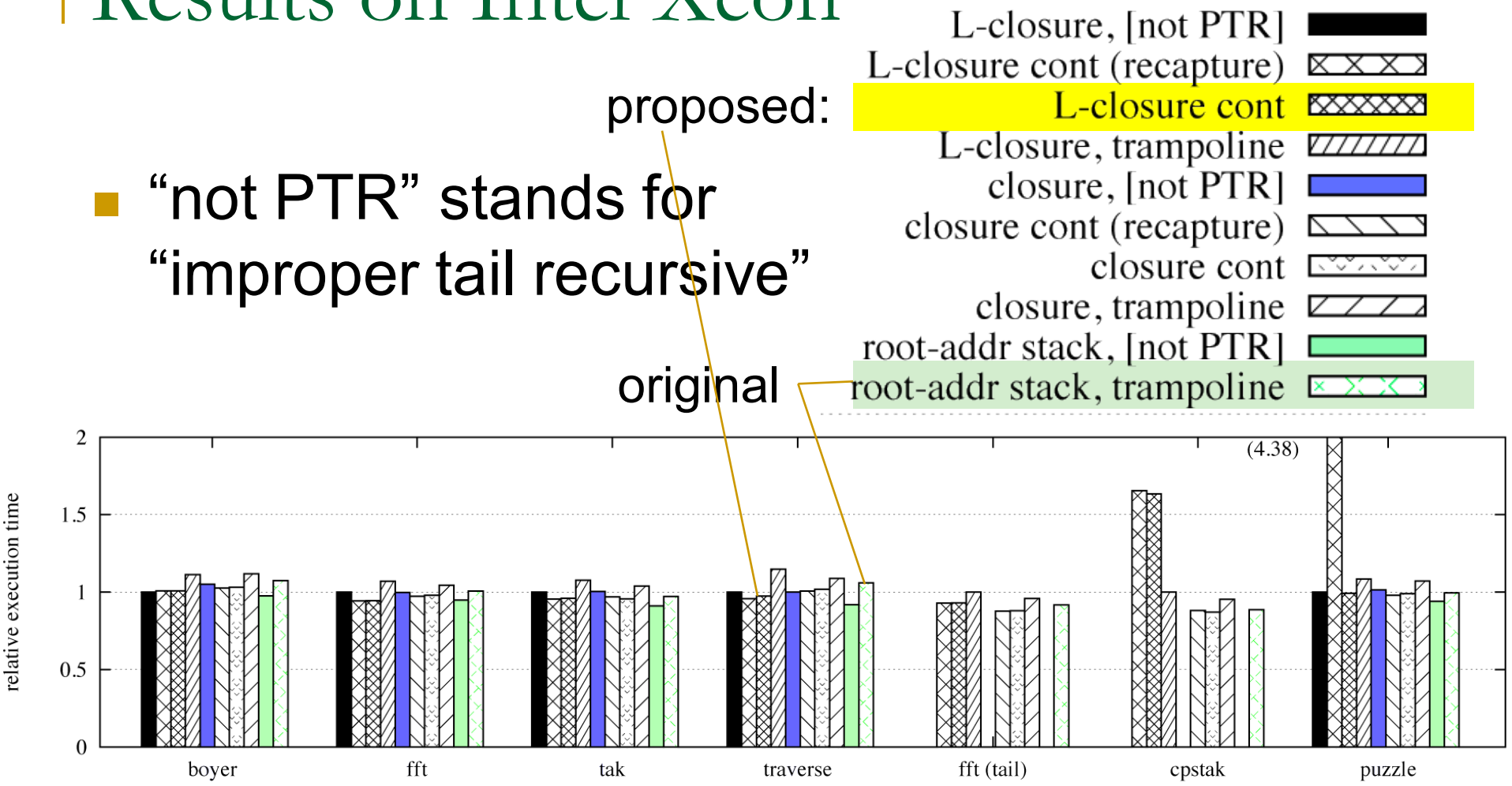
Results on SPARC

- "not PTR" stands for "improper tail recursive"



Results on Intel Xeon

- "not PTR" stands for "improper tail recursive"



発表内容

- (安全な)計算状態操作機構L-closureとは?
  - AT との関係
- これまでの応用例
- これまでの実装
- 最近の実装
- 最近の応用例
- これからの課題

これからの課題

- 短期的
  - 償却時間(amortized time)を下げる実装モデル
  - GCC バージョン4 ベースへ
  - 他言語での実現
- 長期的
  - 信頼性向上: 計算状態の保存と復元
  - 計算状態の保全と再利用に基づく新しい並列実行モデル
  - ソースレベルでの計算状態表現技術

本年度の発表

- 八杉昌宏, 小島啓史, 小宮常康, 平石拓, 馬谷誠二, 湯淺太一. L-Closure を用いた真に末尾再帰的なScheme インタプリタ. 情報処理学会論文誌　プログラミング. (採録決定)
- 平石拓, 八杉昌宏, 湯淺太一. SC 言語処理系における変形規則の再利用機構. コンピュータソフトウェア. (コメント付採録)
- 八杉昌宏. L-Closure：安全な計算状態操作機構. 情報処理, Vol. 51, No. 7, p. 885, July 2010.
- 平石 拓, 河野 卓矢, 八杉 昌宏, 馬谷 誠二, 湯淺 太一. バックトラックに基づく負荷分散の高並列環境における評価, 情報処理学会研究報告 (SWoPP'10).
- Tasuku Hiraishi, Masahiro Yasugi,Takuya Kouno, Seiji Umatani, and Taiichi Yuasa. Tascell: a Backtracking-based Load Balancing Framework, ICS '10 poster presentation.
- Masahiro Yasugi, Tsuneyasu Komiya, Tasuku Hiraishi, and Seiji Umatani, Managing Continuations for Proper Tail Recursion, Proc. of International Lisp Conference 2010, Oct. 2010
- Masahiro Yasugi, Tasuku Hiraishi, Seiji Umatani, and Taiichi Yuasa, Dynamic Graph Traversals for Concurrent Rewriting using Work-Stealing Frameworks for Multicore Platforms, Proc. of IEEE International Conference on Parallel and Distributed Systems (ICPADS 2010), Dec. 2010 (to appear).