

Demonstrační cvičení IFJ
Implementace interpretu IFJ13

18. 11. 2013

Ondřej Navrátil
Zbyněk Křivka

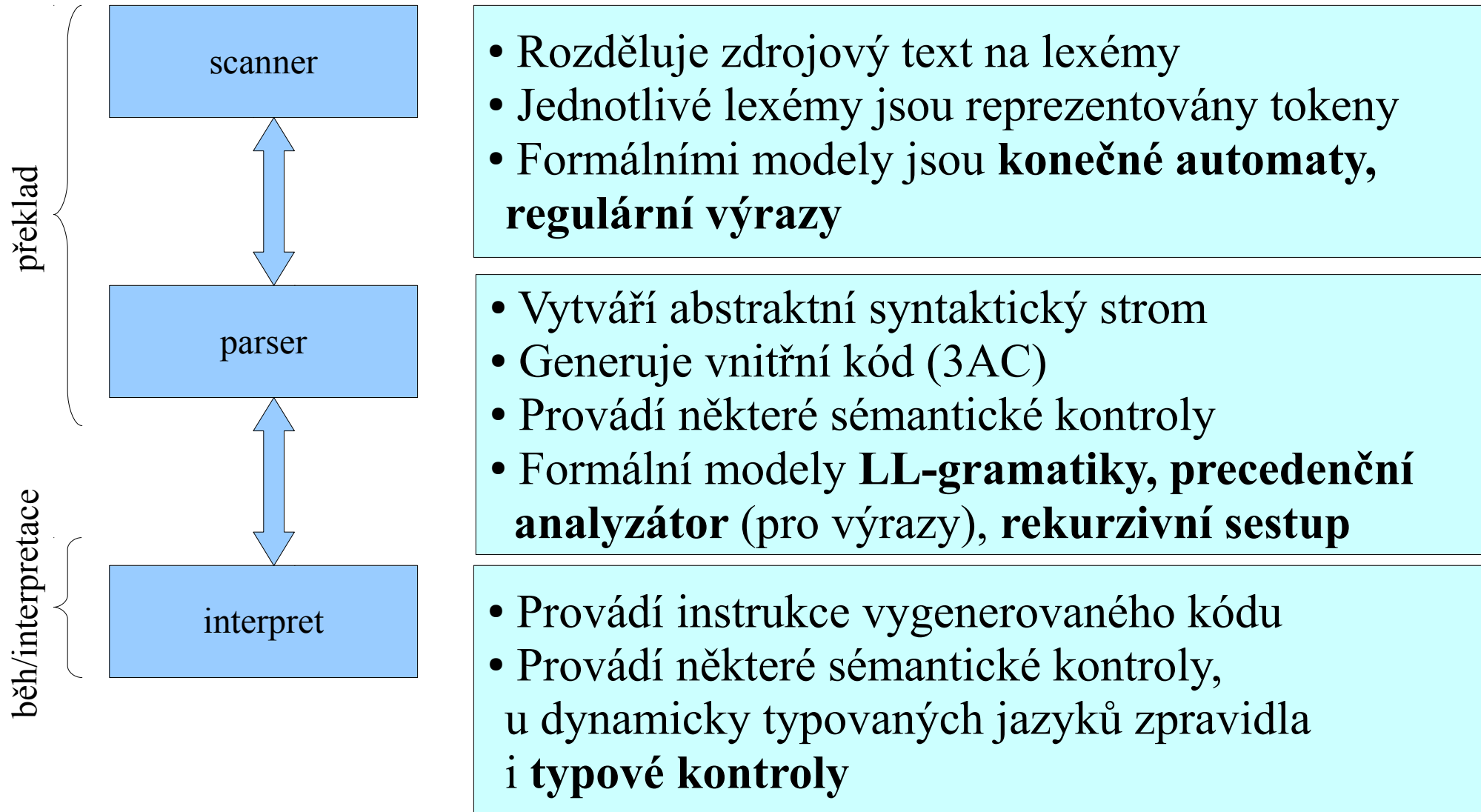
Osnova cvičení

- přehled komponent interpretu
- tvorba identifikátorů, komunikace scanner-parser
- tabulka symbolů, metody implementace
- generování 3AC
- metody SA a jejich kombinace

Úvod

- Povinné části projektu (viz Zadání)
 - Specifikace IFJ13
 - návratové kódy
 - Závazné metody:
 - prediktivní SA/rekurzivní sestup
 - precedenční SA pro výrazy
 - Formální požadavky:
 - způsob odevzdání (obsah archivu, jména souborů, ...)
- Zde probrané možnosti nejsou závazné, není-li řečeno jinak, a existují i další správná řešení!

Schéma interpretu



Tabulka symbolů

- Scanner ze své podstaty nepotřebuje TS
- Pokud předává odkaz, musí parser provádět další kontroly

```
function novaFunkce ($parametr) ...  
$a = dalsiFunkce ();
```

V obou případech rozezná scanner id funkce.

- Má však scanner vkládat, nebo vyhledávat v TS?
- Jak se zachovat při duplicitě/chybějícím záznamu v TS?
- U proměnných se musí parser starat o úroveň rozsahu platnosti.

Implementace TS pro překlad

- Tabulka pro funkce (globální) – kontrola **vícenásobných definic**
- Definice proměnných se kontrolují **až za běhu**, při překladu tedy pro ně tabulku nepotřebujete

```
if ($podminka)
    $a = 10;
else
    $b = 10;

put_string($b);
```

Implementace TS pro interpretaci

Informace o identifikátorech (jméno, druh, typ):

- Tabulka funkcí obsahuje navíc odkazy na 3AC s tělem funkce
- Tabulka proměnných musí respektovat jejich rozsah platnosti, rekurzivní volání

Hodnoty proměnných (data, inicializovanost):

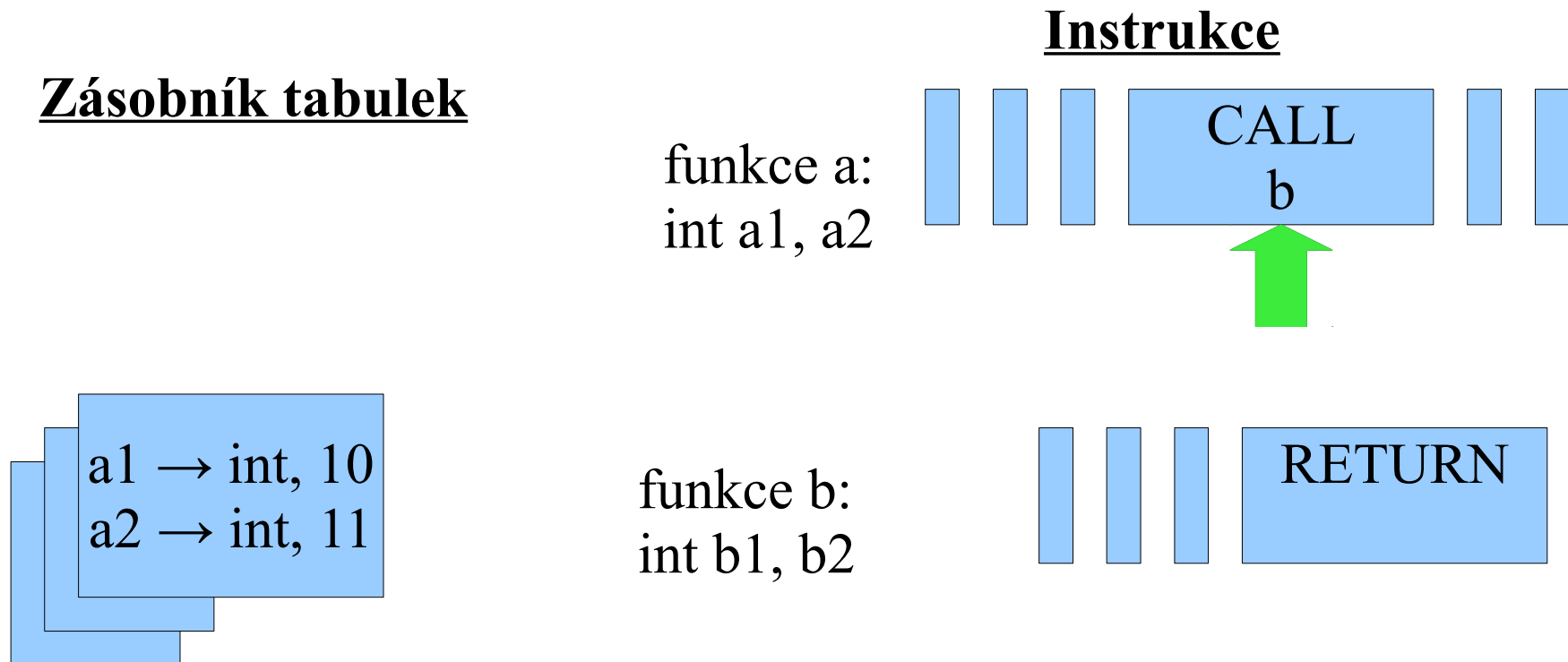
- Dynamicky versus staticky typované jazyky

Implementace:

- A) Vytváření rámců (např. x86) – vyžaduje **hlubší analýzu v SA**, obtížnější obsluha, ale je efektivnější
- B) Lze použít i abstraktnější **zásobník tabulek** pro jednotlivé úrovně
 - Nutno vyřešit rozhraní pro volání funkce a návrat hodnoty

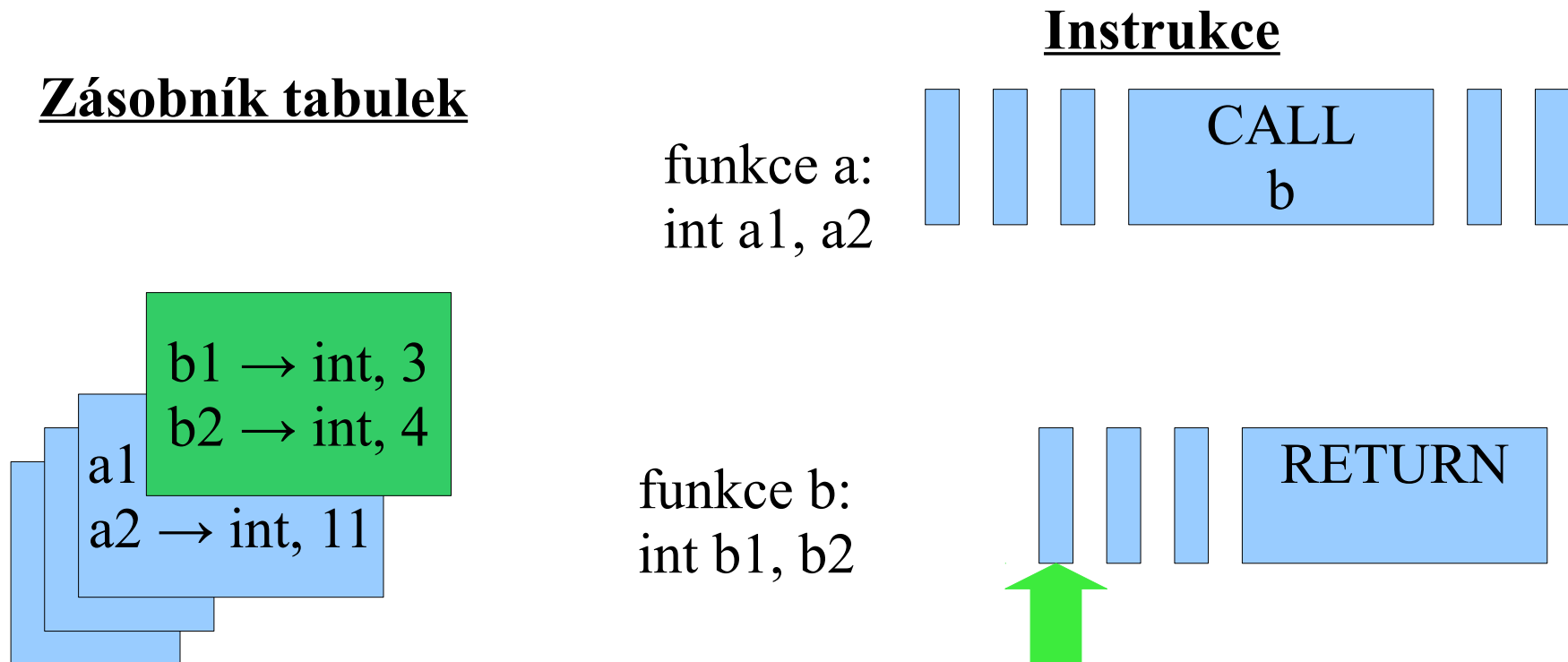
Zásobník TS

- Při volání funkce vytvoříme novou TS, vložíme na vrchol
- Hledání proměnné se provádí v tabulce na vrcholu
- Při návratu z funkce získáme původní kontext odstraněním tabulky na vrcholu



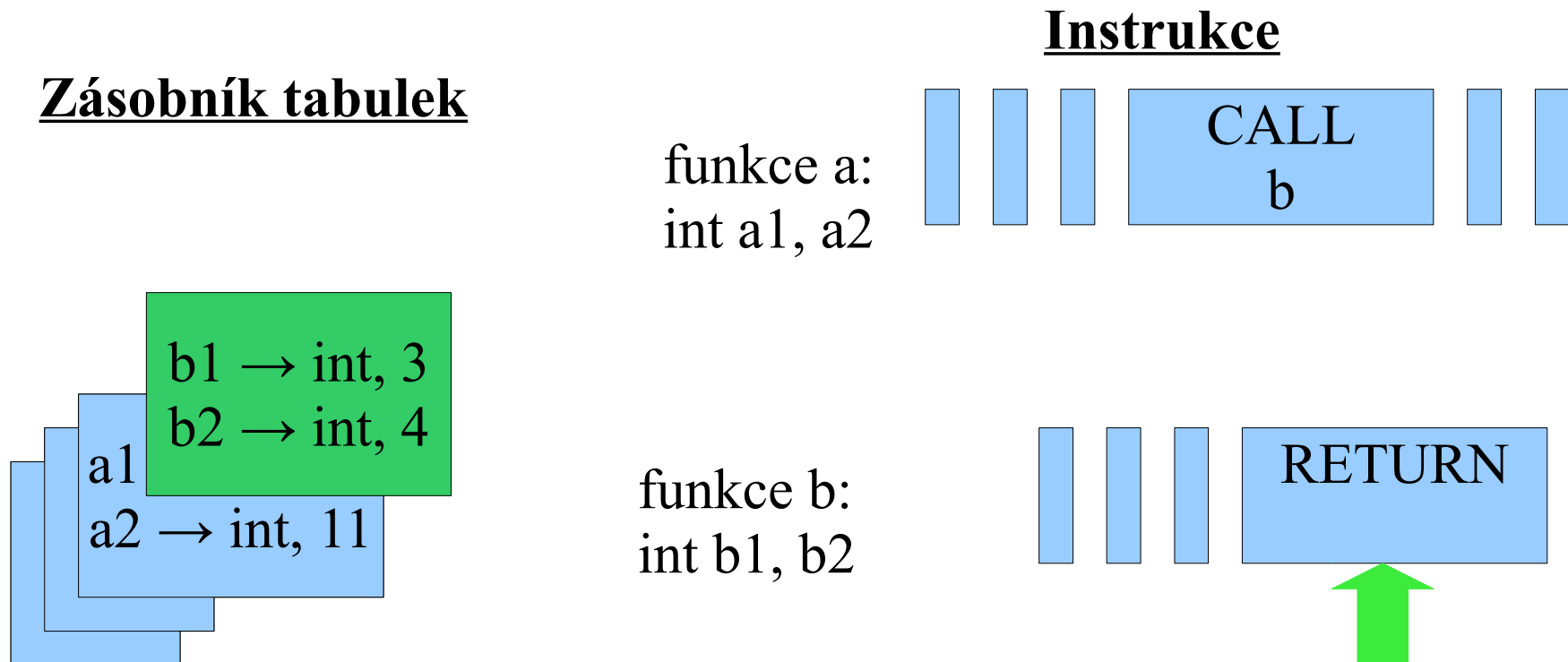
Zásobník TS

- Při volání funkce vytvoříme novou TS, vložíme na vrchol
- Hledání proměnné se provádí v tabulce na vrcholu
- Při návratu z funkce získáme původní kontext odstraněním tabulky na vrcholu



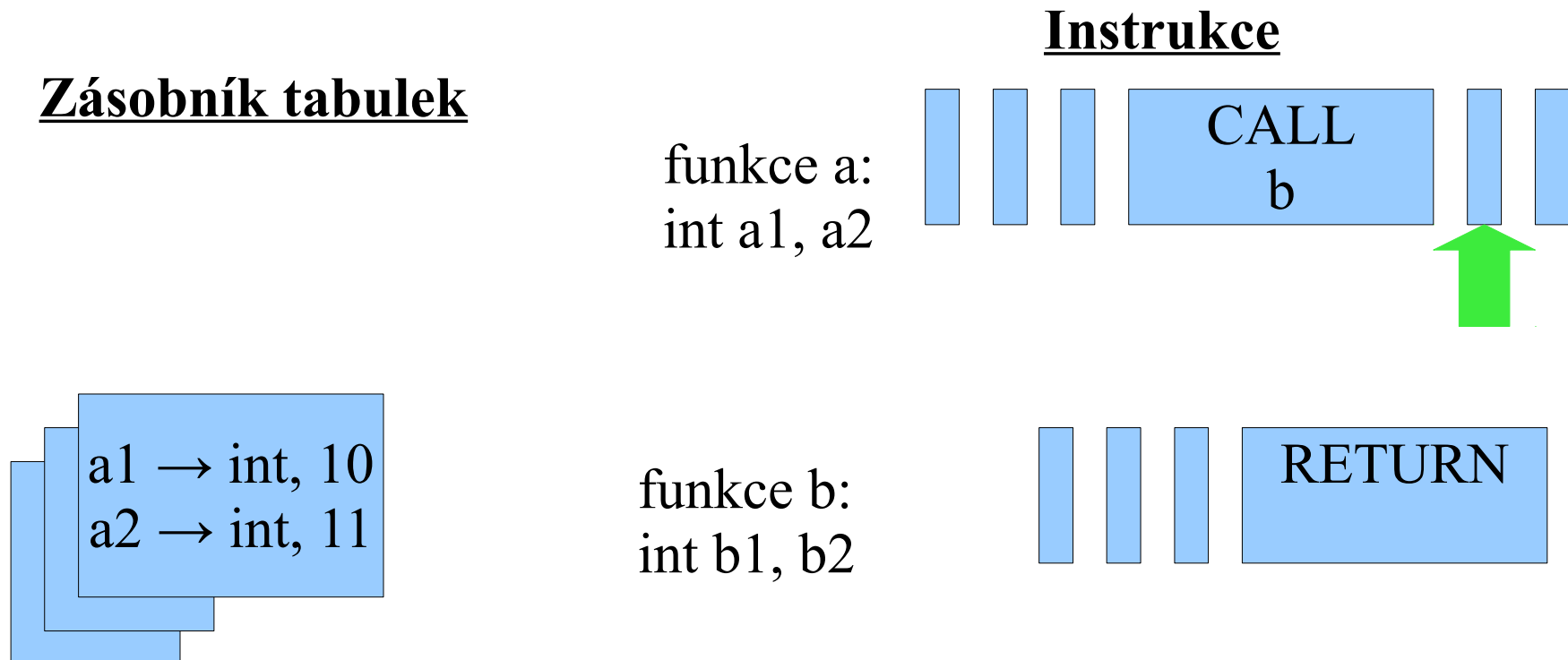
Zásobník TS

- Při volání funkce vytvoříme novou TS, vložíme na vrchol
- Hledání proměnné se provádí v tabulce na vrcholu
- Při návratu z funkce získáme původní kontext odstraněním tabulky na vrcholu



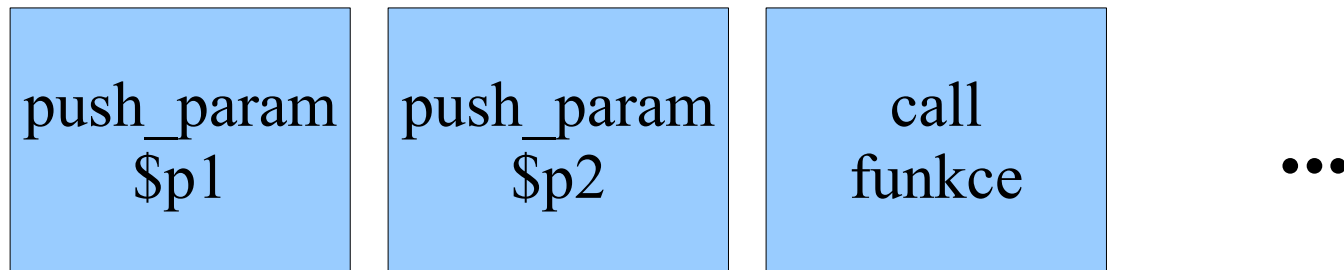
Zásobník TS

- Při volání funkce vytvoříme novou TS, vložíme na vrchol
- Hledání proměnné se provádí v tabulce na vrcholu
- Při návratu z funkce získáme původní kontext odstraněním tabulky na vrcholu



3AC volání funkce

$\$x = \text{funkce}(\$p1, \$p2);$



- **push_param** – ukládání parametrů (do TS, příp. jina)
- **call** – nová TS na vrchol zásobníku, kontrola počtu a naplnění parametrů, zapamatování bodu návratu (možno v TS), vyhledání funkce a skok na tělo funkce

3AC návratu

```
return 10;
```



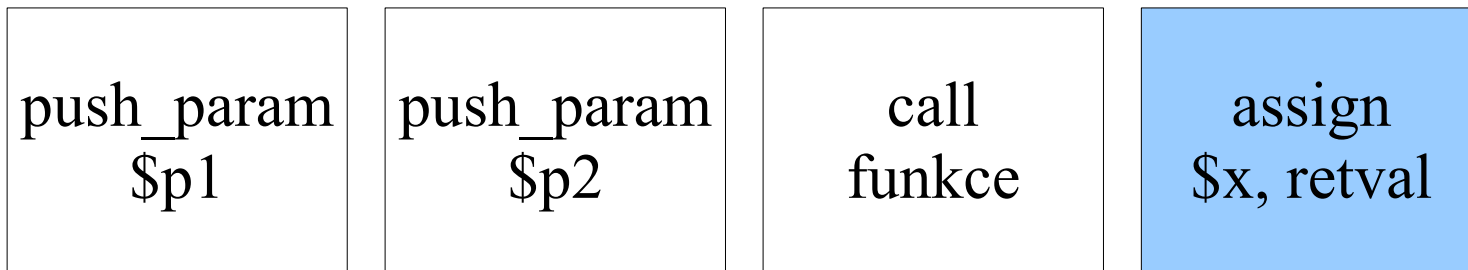
assign_prev
retval, 10

return

- **assign_prev** – přiřazení do retval (v nadřazené TS)
- **return** – návrat na zapamatovanou instrukci, odstranění nejvrchnější TS
- **Pozor, pokud jsme na globální úrovni!**

3AC volání funkce

\$x = funkce (\$p1, \$p2) ;



- **push_param** – ukládání parametrů (do TS, příp. jina)
- **call** – nová TS na vrchol zásobníku, kontrola počtu a naplnění parametrů, zapamatování bodu návratu (možno v TS), vyhledání funkce a skok na tělo funkce
- **assign** – přiřazení návratové hodnoty (nutno uchovat po **returnu**, opět možno v TS)

Syntaktická analýza

- Doporučenou metodou je **rekurzivní sestup (RS)** v kombinaci s **precedenční syntaktickou analýzou (PSA)** pro výrazy
- Nutno zajistit korektní předávání řízení mezi oběma metodami

```
while ( $a > 5 )
```

- Po přijetí tokenu '(' lze jednoznačně spustit **PSA**

```
$a = funkce();  
$a = $a - 1;
```

- Po přijetí tokenu '=' nevíme, jestli nenásleduje volání funkce
- Token načtený navíc je **nutno předat PSA**
- Zatím jsme neřešili, jak proběhne předání řízení zpět **RS**

Ukončení precedenční SA

```
$a = $a - 1;
```

- Token ';' není v abecedě precedenční SA, můžeme jej tedy považovat za konec vstupu pro precedenční SA – token '\$'

```
while ( $a > 5 )
```


- U ')' nevíme, zda-li jej brát jako pokračování precedenční SA, nebo ukončení závorek u **while**
- Budeme si tudíž muset formální model trochu "ohnout"

Úprava precedenční SA

- Detekujeme tuto situaci v tabulce

Terminál na vrcholu zásobníku

		Vstupní token					
		+	*	()	<i>i</i>	\$
+	>	<	<	>	<	>	
*	>	>	<	>	<	>	
(<	<	<	=	<		
)	>	>		>		>	
<i>i</i>	>	>		>		>	
\$	<	<	<		<		



- Dále je třeba situaci **ošetřit** úpravou algoritmu
- Například v tomto případě ukončit precedenční SA a případně **vrátit token ')' zpět na vstup**

Další dotazy

- Zotavení z chyb – návratový kód první chyby!
- Implicitní konverze ve výrazech – viz zadání!
- Nástroj pro tvorbu LL tabulky
- Mírně zastaralé "Jak na projekt" z roku 2008
-
-
-
- ... čtete zadání, wiki, fórum ...