

Διαδικασίες, η Στοίβα, και οι
Συμβάσεις Χρήσης/Διατήρησης/Επαναφοράς
Καταχωρητών

06α (§6.1-6.4) – 11-14 Μαρτίου 2022 – Μανόλης Κατεβαίνης

Procedure Activation Frame: πού στη μνήμη;

- Activation Frame: τοπικές μεταβλητές & δεδομένα Διαδ.
 - μπορεί και πίνακες, ενδεχομένως μεταβλητού μεγέθους
 - Ενεργές («ζωντανές») μόνο όσο η ίδια η διαδικασία «ενεργή»
 - ενεργή και όσο περιμένει να επιστρέψουν παιδιά της
 - «Πεθαίνουν» (deallocated) μόλις επιστρέψει η διαδικασία (LIFO)
- Πού στη μνήμη; – όχι σε χώρο ανά (στατ.) διαδικασία
 - «στατικά ορισμ.» διαδ. συχνά περισσ. από «δυναμικά» ενεργές
 - εάν ένας ανά διαδικασία, δεν υποστηρίζει αναδρομή
- Πού στη μνήμη; → Στοιίβα (“Runtime Stack”) – LIFO call/ret
 - μόνον οι («δυναμικά») ενεργές διαδικασίες πιάνουν χώρο
 - υποστηρίζει Αναδρομή (recursion) (άμεση ή και κυκλική)

Τοπικές Μεταβλητές (& Ορίσματα) σε Καταχωρητές

- Συνήθως λίγες, και συνήθως χρησιμοποιούνται συχνά
 - μιλάμε για τοπικές βαθμωτές μεταβλητές – όχι για τοπ. πίνακες
 - ομοίως και τα ορίσματα (arguments) της διαδικασίας
 - η C δέχεται μόνον βαθμωτά ορίσματα – πίνακες μόνο μέσω ptr.
- ⇒ Τα θέλουμε σε Καταχωρητές: Ταχύτητα, Συντομία εντολ.
- Αλλά οι καταχωρητές είναι ένα set, για όλες τις διαδικ.!
 - ενώ η κάθε διαδικασία έχει τις δικές της τοπ. μτβλ. & ορίσματα
- ⇒ Πρέπει σε κάθε κλήση/επιστρ. να σώζουμε/επαναφ.
 - στο Activation Frame της διαδικασίας, στη Στοιίβα
 - τίνος ευθύνη; Του Καλούντα (Caller) ή του Καλουμένου (Callee)?

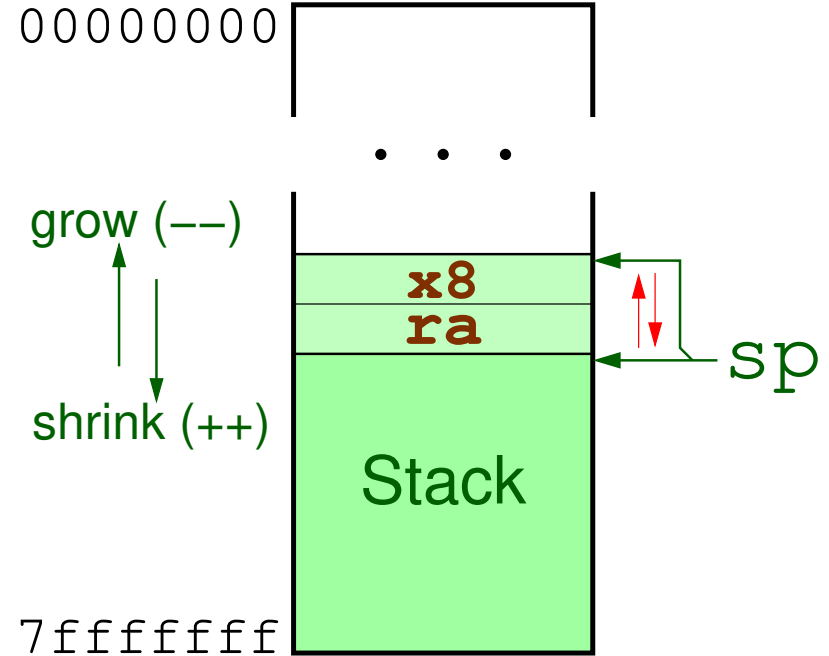
Σώσιμο Καταχωρητών στη Στοίβα και Επαναφορά

- Σώσιμο (save) των **ra**, **x8**:

```
addi sp, sp, -8
sw ra, 4(sp)
sw x8, 0(sp)
```

- Επαναφορά (restore):

```
lw ra, 4(sp)
lw x8, 0(sp)
addi sp, sp, +8
```



- Ο stack pointer (sp) πρέπει να δείχνει πάντα στο τελευταίο (minimum address) κατειλημμένο Byte στη στοίβα
⇒ στοίβα αυξάνει πριν το πρώτο save, μικραίνει μετά το τελευταίο

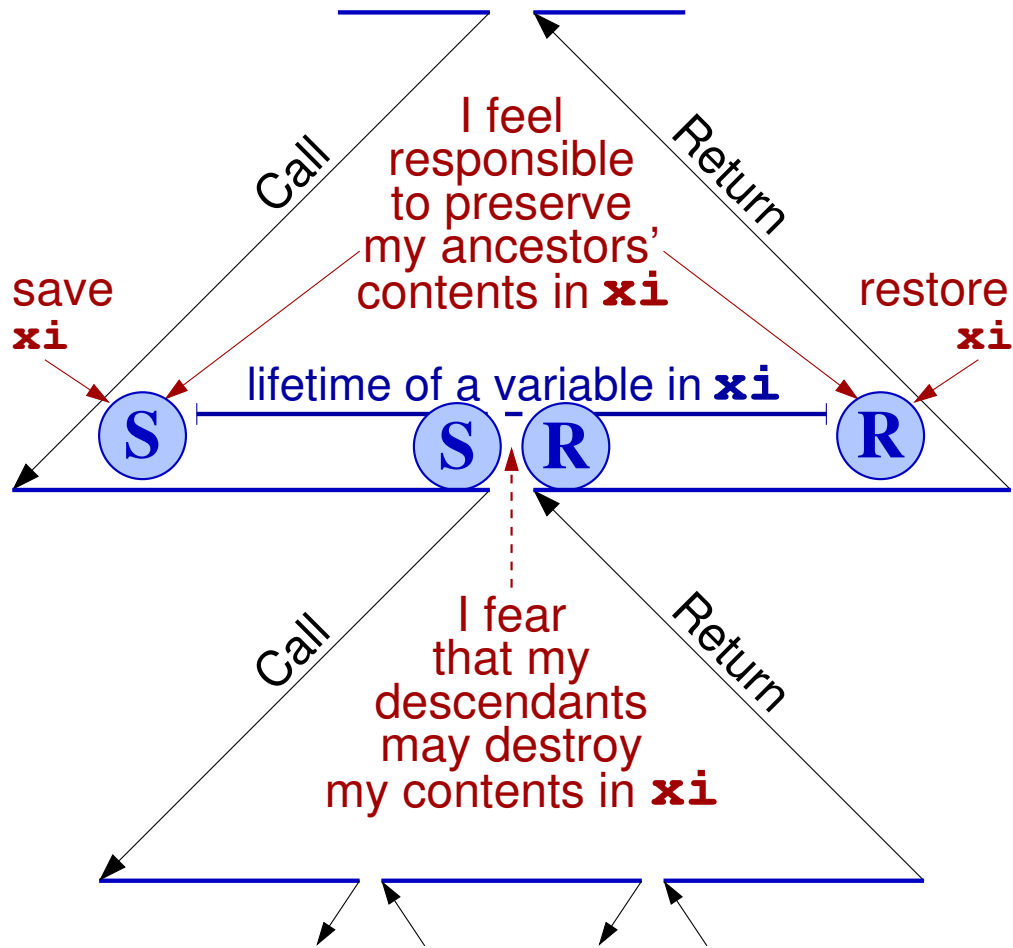
Χωριστή Μετάφραση: δεν ξέρω γονέα/παιδιά μου!

- Θέλουμε οι διαδικασίες σε χωριστά αρχεία
 - ⇒ Όταν μεταφράζουμε (Compile) μιά διαδικασία, δεν ξέρουμε τη χρήση καταχωρητών του γονέα μου (της διαδικασίας που με κάλεσε), ούτε τη χρήση καταχωρητών των ενδεχομένων παιδιών μου (των διαδικασιών που εγώ ενδεχομένως θα καλέσω)
 - ⇒ Ανάγκη Σύμβασης Κλήσης Διαδικασιών
 - έχει οριστεί ενιαία, για όλους τους Compilers του Ρεπ. RISC-V
 - εγγυάται την αρμονική συνένωση (linking) αρχείων δυαδικού κώδικα (π.χ. βιβλιοθηκών) παλαιών/νέων & ανά τον κόσμο

«Διάρκεια Ζωής» (Lifetime) Μεταβλητής/Καταχωρητή

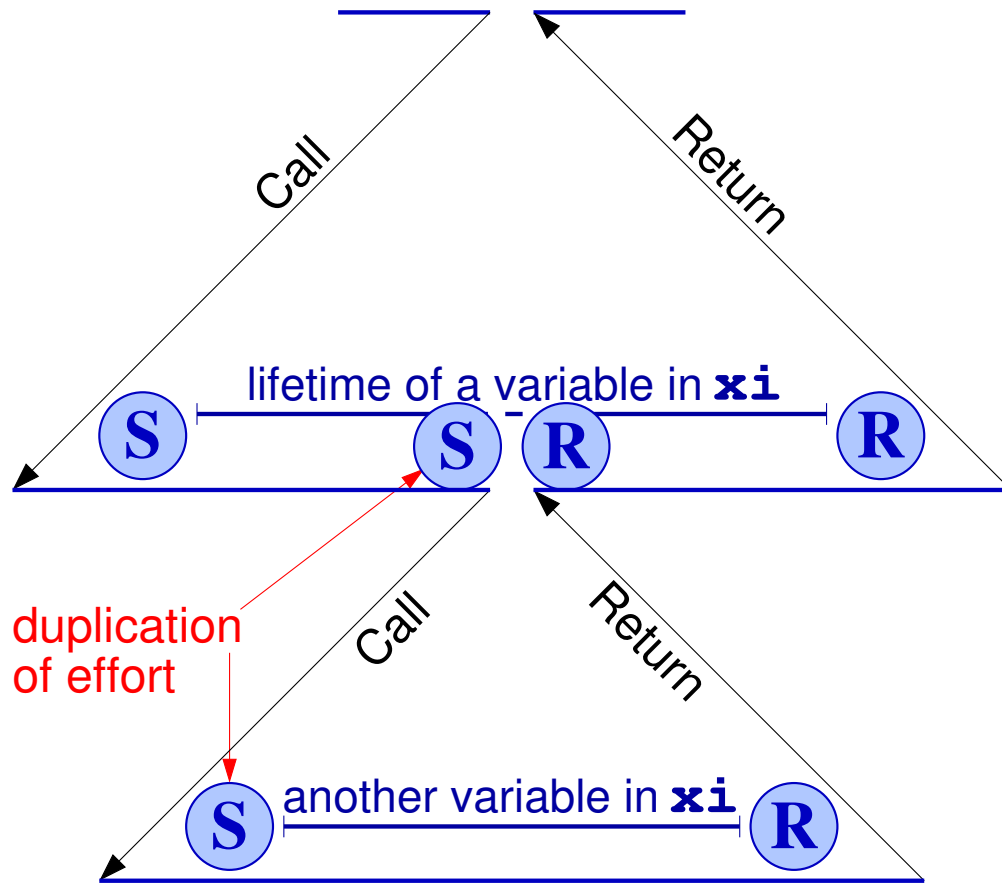
- Στη διάρκεια της ζωής «τον χρειαζόμαστε» (την τιμή του)
- Όταν «νεκρός», δεν τον χρειαζόμαστε (τιμή = σκουπίδια)
- Διάρκεια ζωής =
- Από την εκχώρηση νέας τιμής σε αυτήν/αυτόν
 - η οποία νέα τιμή δεν υπολογίζεται βάσει της παλαιάς του τιμής από την ίδια την εντολή που του εκχωρεί τη νέα τιμή
- Μέχρι την τελευταία ανάγνωση της τιμής του πριν την επόμενη εκχώρηση (ανεξάρτητης) νέας τιμής
 - εντολές που διαβάζουν και γράφουν τον ίδιο καταχωρητή (π.χ. `addi t0, t0, 1`) ούτε ξεκινούν ούτε τερματίζουν μία «ζωή»

Συντηρητική προσέγγιση: προστασία πανταχόθεν



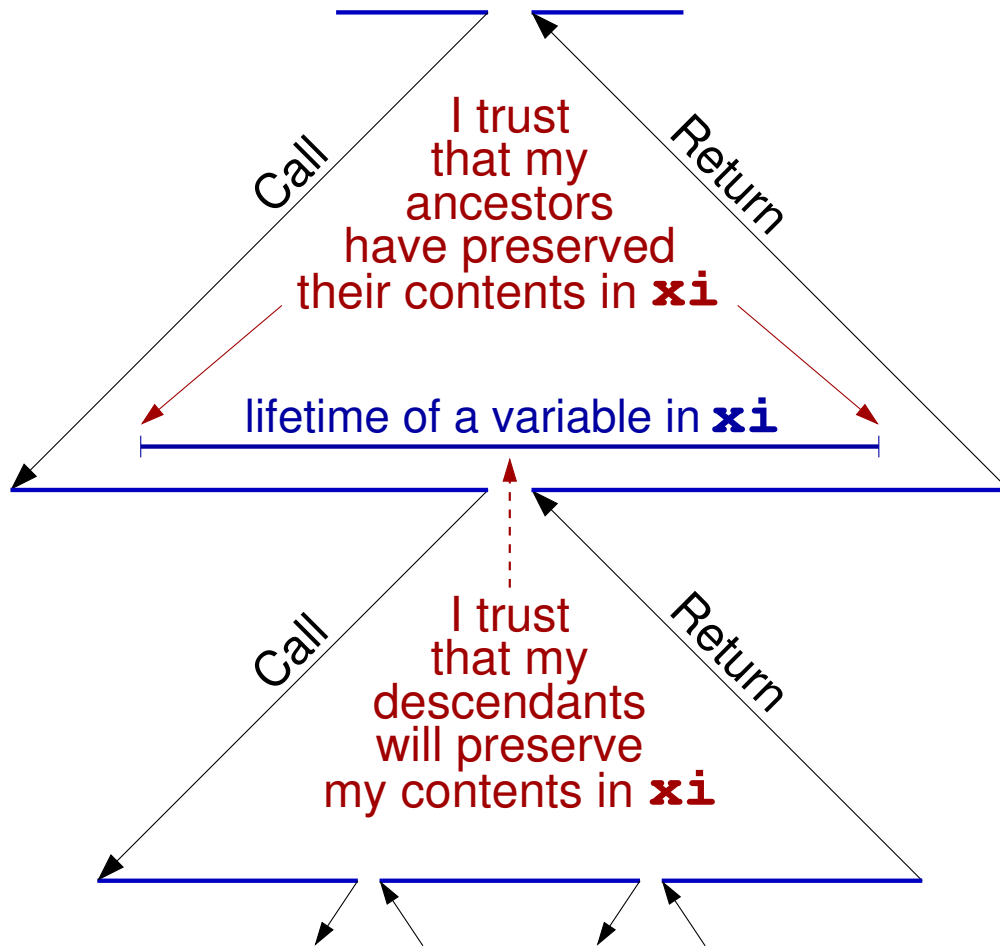
- Χρόνος εκτέλεσης οριζόντια, κλήσεις κατακόρυφα
- Πριν βάλω κάτι δικό μου στον καταχωρητή \mathbf{xi} , σώζω τα παλαιά περιεχόμενά του στη στοίβα μου, και τα επαναφέρω όταν τελειώσω, πριν επιστρέψω
- Πριν καλέσω παιδί, προστατεύω ό,τι έχω στον \mathbf{xi}
- Χρειάζονται όλα αυτά;

Περισσότερη δαπανηρή η προστασία πανταχόθεν



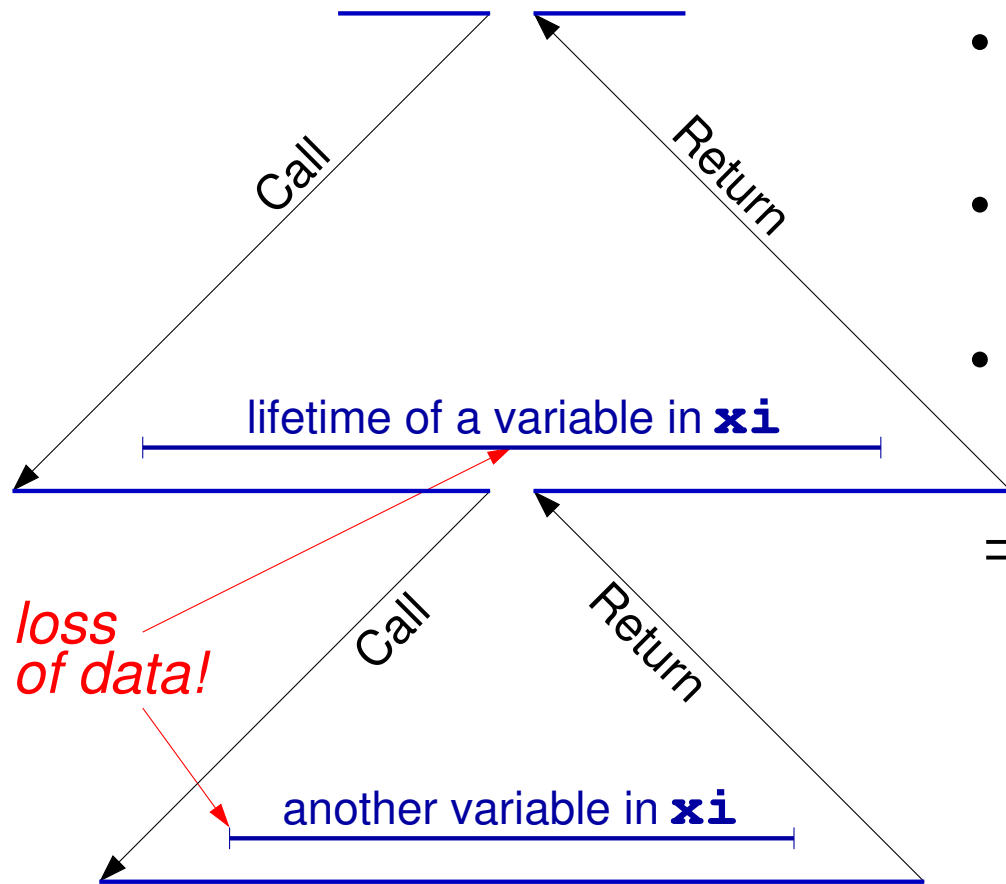
- Και ο γονέας φοβάται μήπως οι απόγονοί του καταστρέψουν το περιεχόμενο του καταχωρητή
- Και ο κάθε απόγονος προστατεύει ό,τι είχαν οι πρόγονοί του μέσα στους καταχωρητές που αυτός χρησιμοποιεί
- Τελικά, καταλήγει να γίνεται διπλή δουλειά...

Χαλαρή προσέγγιση: ας προσέξουν οι άλλοι



- Ελπίζω οι πρόγονοί μου να ήταν σώφρονες
- Ελπίζω οι απόγονοί μου να είναι υπεύθυνοι
- Εγώ, πάντως, είμαι χαλαρός και ανεύθυνος...
- Η υπερβολική χαλαρότης βλάπτει σοβαρά την υγεία!

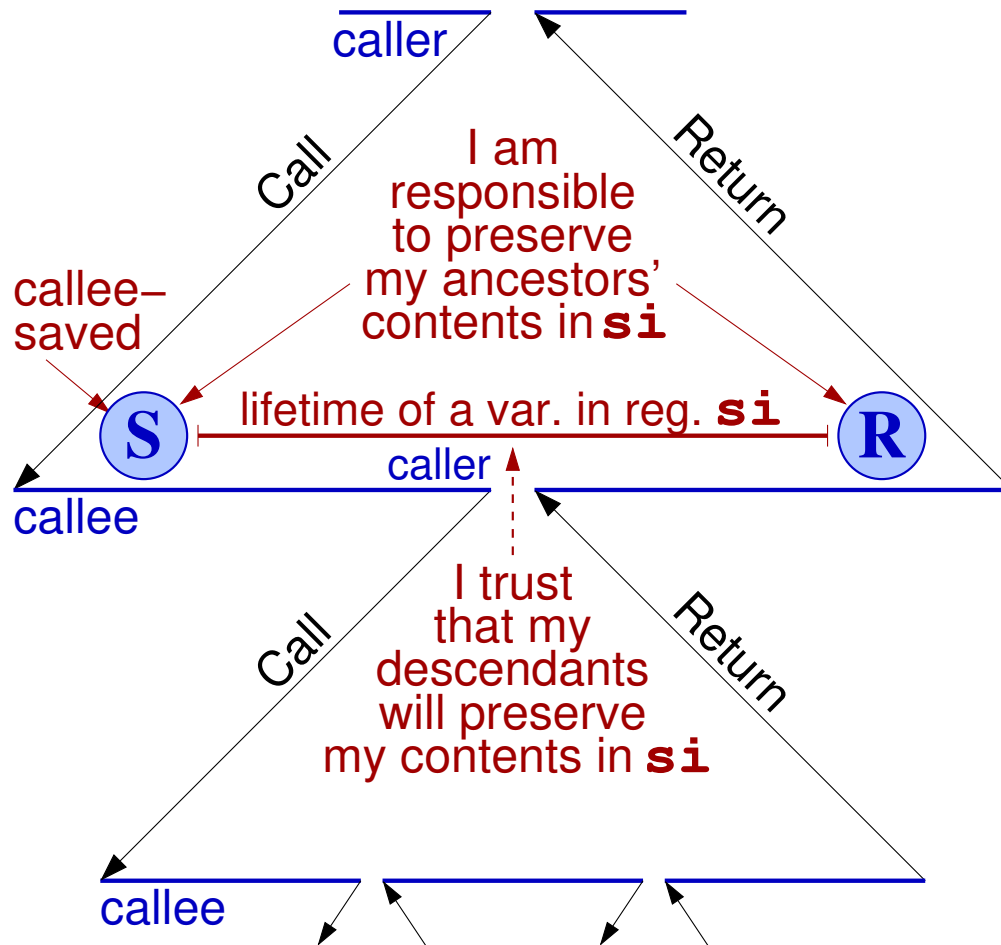
Καταστροφικό το να τα περιμένω όλα από τους άλλους



- Εγώ τα περιμένω όλα από τους άλλους
- Και οι άλλοι τα περιμένουν όλα από εμένα
- Άρα κανείς δεν κάνει τίποτα, και όλοι χάνουμε τα πάντα...

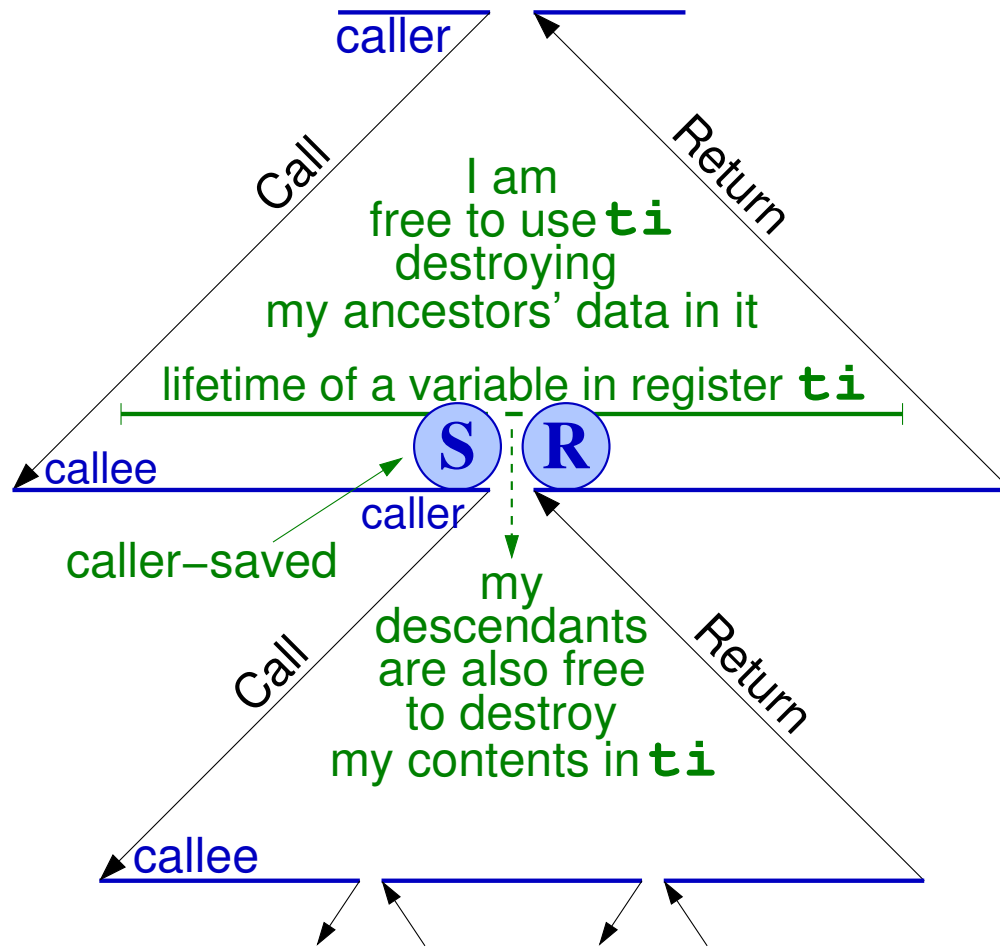
⇒ Πρέπει να επιλέξουμε έναν από τους δύο, καλούμενο ή καλούντα, ως υπεύθυνο για save/restore: Δύο δυνατές συμβάσεις/πολιτικές

Επιλογή 1: Callee-Saved (Saved registers)



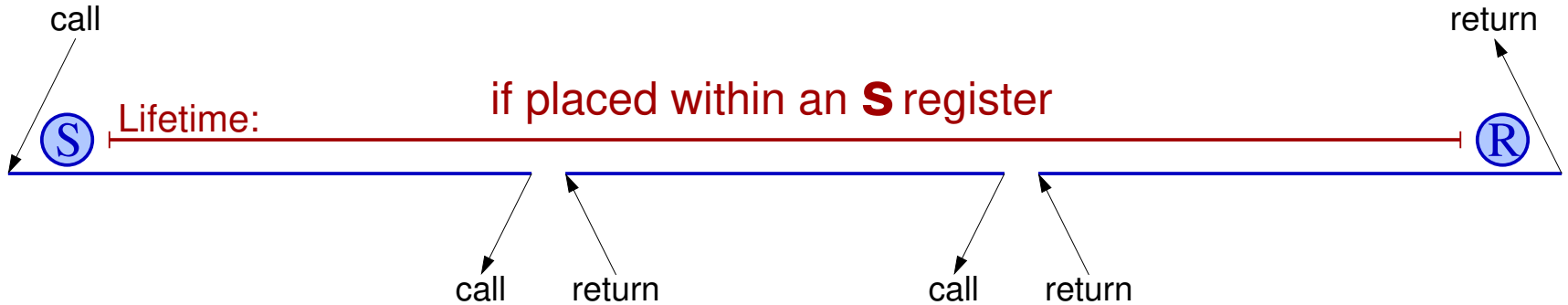
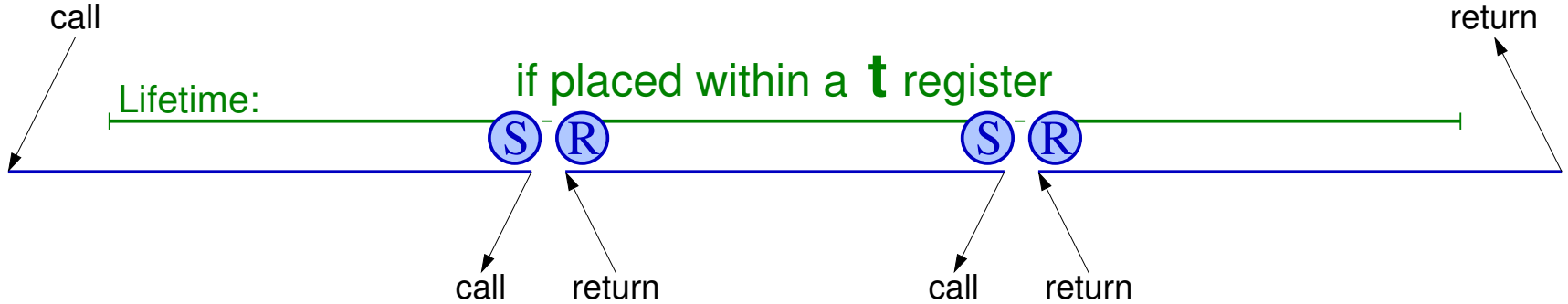
- Ευθύνη του Καλουμένου να διατηρήσει ό,τι περιεχόμενα είχαν οι κατάχωρητές που αυτός θα χρησιμοποιήσει
- Όταν εγώ καλώ, ξέρω ότι οι απόγονοί μου θα διατηρήσουν τα δικά μου περιεχόμενα.
- Register contents *saved* across procedure calls
- Σκεπτικό: αναβάλω το σώσιμο, ελπίζοντας κανείς απόγονος να μη χρειαστεί αυτόν τον καταχωρητή

Επιλογή 2: Caller-Saved (Temporary registers)



- Ευθύνη του Καλούντα να διατηρήσει ό,τι έχουν οι κατάχωρητες όταν αυτός καλεί παιδιά
- Όταν εμένα με καλούν, εγώ είμαι ελεύθερος να τους χρησιμοποιήσω
- Reg's for *temporary* values, *not* preserved across calls
- Σκεπτικό: ελπίζω οι πρόγονοί μου να μην είχαν κάτι χρήσιμο σε αυτόν τον κατάχωρητή

Lifetimes of variables that span 2 or more procedure Calls

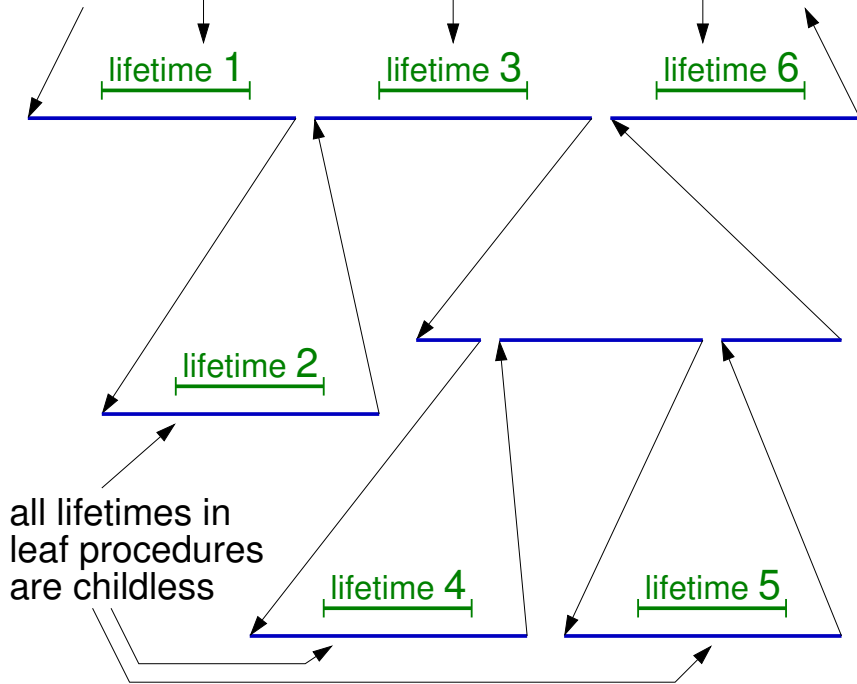


"s" (saved) register is preferable: fewer save–restores to stack

Lifetimes of variables that contain no procedure Calls

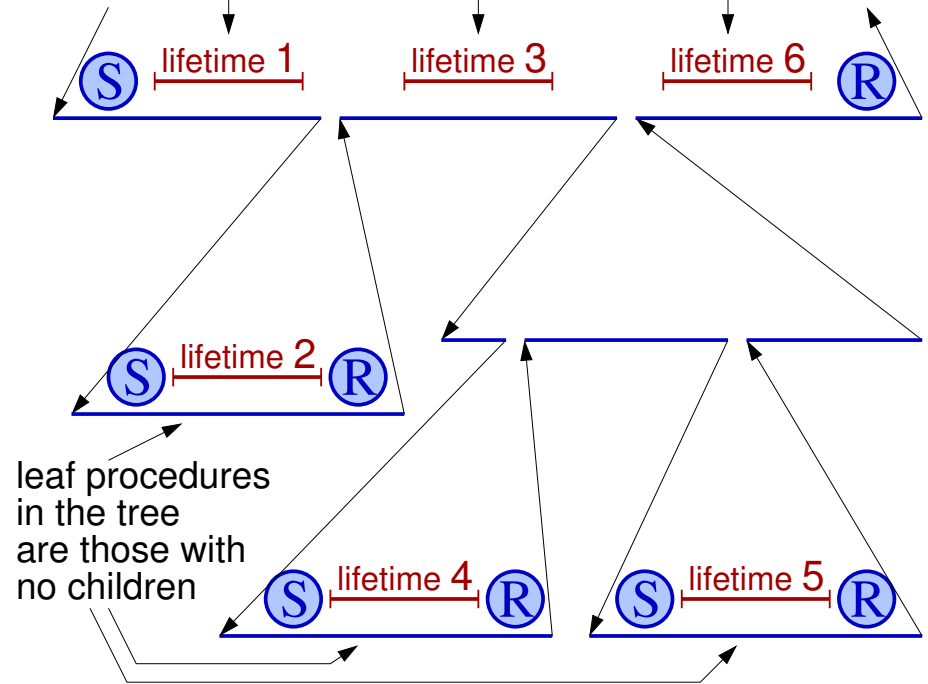
if placed within **t** registers:

three independent lifetimes (variables):
no data need to be preserved from one to the other



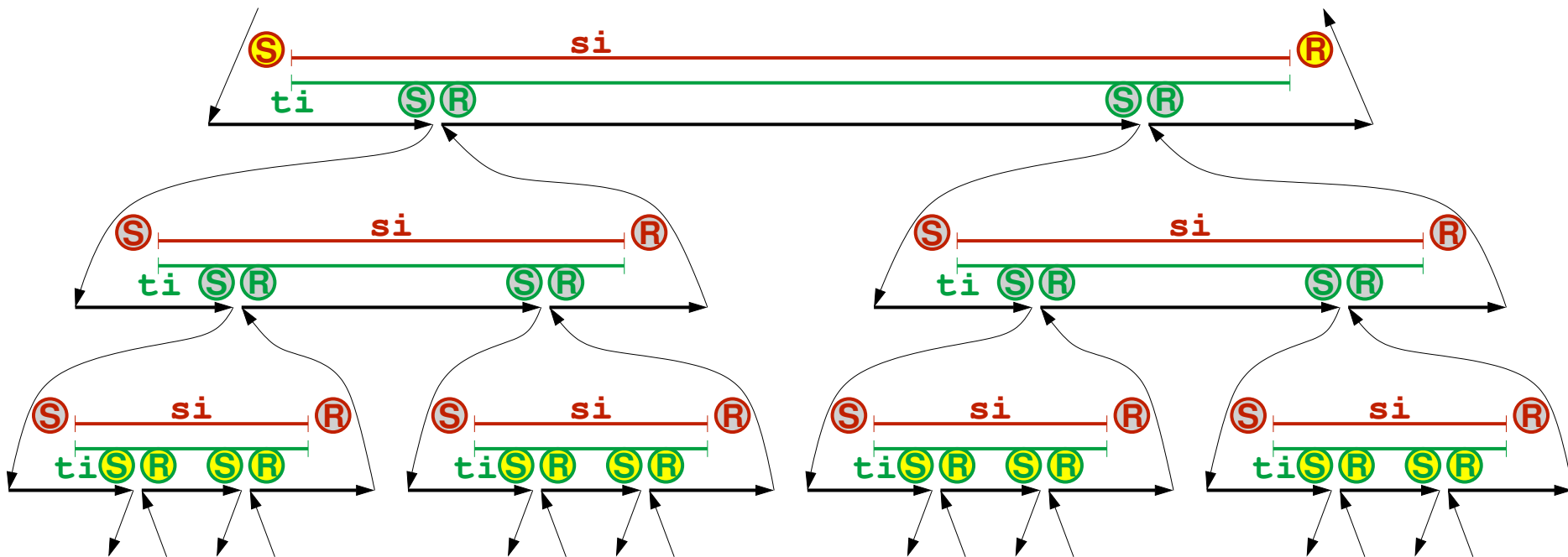
if placed within **S** registers:

three independent lifetimes (variables)
within one, same s register



"t" registers are preferable for childless lifetimes: no save-restore to stack 14

Τι διαφορά κάνει αν θα σώσει ο ένας ή ο άλλος;



- Για ό,τι δεδομένα πρέπει να διατηρηθούν «ζωντανά» διαμέσου κλήσεων, είναι το ίδιο αν θα τα σώσει ο πρόγονος (**ti**) ή ο απόγονος (**si**) στα ενδιάμεσα επίπεδα κλήσεων (γκρί S-R), εκτός στο πάνω-πάνω και στο κάτω-κάτω επίπεδο (**κίτρινα** S-R)
- Στο κατώτατο επίπεδο, δεν ξέρω εάν οι απόγονοί μου θα τον χρειαστούν τον καταχ., άρα αναβάλω το σώσιμο (μέσω **si**) ώστε εκείνοι να κάνουν ό,τι ξέρουν & συμφέρει

Δύο σύνολα reg's – ένα για φύλλα, άλλο για μακρόζωες

- Σαν να έχω χωρίσει τους καταχωρητές σε δύο υποσύνολα: Οι μεν **ti** για όλες τις διαδικασίες-φύλλα και τις ζωές χωρίς παιδιά, οι δε **si** στις «εσωτερικές» διαδικασίες του δένδρου για τις «μακρόβιες» μεταβλητές –εκείνες με πολλαπλές κλήσεις παιδιών στη διάρκεια της ζωής τους
- Εάν η κάθε διαδικασία που καλεί παιδιά καλεί κατά μέσον όρο π.χ. 10 παιδιά, είναι σαν το δένδρο καλεσμάτων να είναι 10-δικό δένδρο, οπότε το 90% των κόμβων του είναι φύλλα \Rightarrow no save/restore's στο 90% των περιπτώσεων!

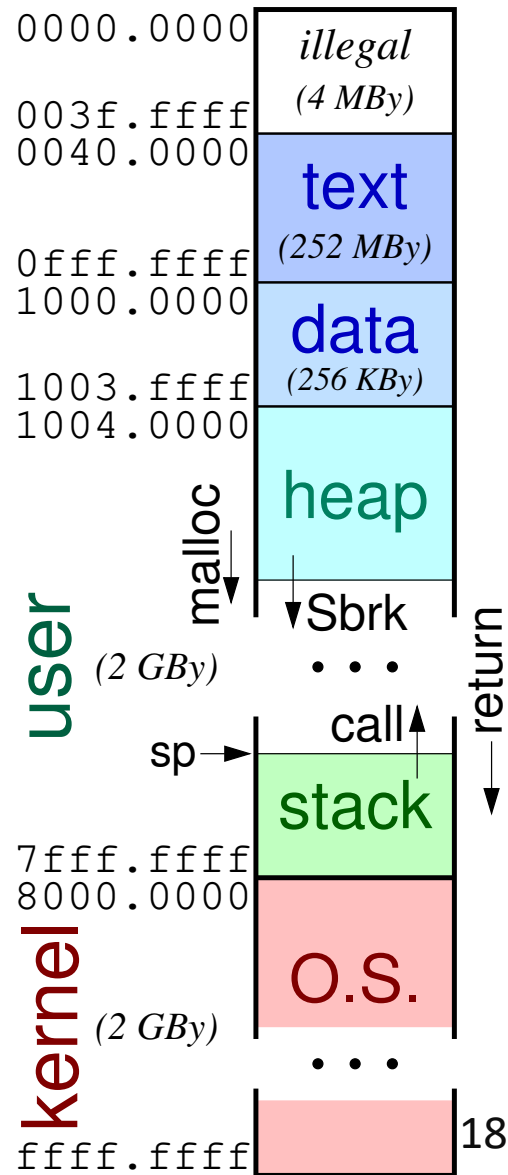
x0	zero	
x1	ra (return address)	"C" sp
x2	sp (stack pointer)	
x3	gp (global pointer)	
x4	tp (thread pointer)	
x5	t0 (caller saved)	
x6	t1 (caller saved)	RV "C" popular
x7	t2 (caller saved)	
x8	s0 / fp (or frame ptr)	
x9	s1 (callee saved)	
x10	a0 (1st arg/ret.val)	
x11	a1 (2nd arg/rv/tmp)	
x12	a2 (3rd arg / tmp)	
x13	a3 (4th arg / tmp)	
x14	a4 (5th arg / tmp)	
x15	a5 (6th arg / tmp)	
x16	a6 (7th arg / tmp)	
x17	a7 (8th arg / tmp)	
x18	s2 (callee saved)	
x19	s3 (callee saved)	
x20	s4 (callee saved)	
x21	s5 (callee saved)	
x22	s6 (callee saved)	
x23	s7 (callee saved)	
x24	s8 (callee saved)	
x25	s9 (callee saved)	
x26	s10 (callee saved)	
x27	s11 (callee saved)	
x28	t3 (caller saved)	
x29	t4 (caller saved)	
x30	t5 (caller saved)	
x31	t6 (caller saved)	

Συμβάσεις Χρήσης Καταχωρητών

- Σύμβαση Κλήσης Διαδικασιών
 - μεταξύ Compilers – δεν αφορά το hardware, δεν την επιβάλλει το hardware
- Τα Ορίσματα ίδια όπως Temporaries
 - Ορίσματα στους **a0**, **a1**, **a2**,...
 - Επιστρεφόμενη τιμή στον **a0** (& **a1** ?)
 - Όσοι καταχωρητές "**a**" (arguments) δεν απασχολούνται από ορίσματα είναι διαθέσιμοι για χρήση ως Temporaries
- Ο *RV32E (embedded)* έχει μόνον 16 καταχ.
- Ο *RVC (compressed)* συμπιέζει όσες εντολές χρησιμοποιούν μόνο τους **x0...2**, **x8...15**

Πώς μεγαλώνουν Heap, Stack: Sbrk, sp

- Stack pointer (sp) δείχνει πάντα την άκρη της κατειλημμένης περιοχής
 - sp-- δηλώνει εμμέσως προς OS: μεγάλωμα
- Για να μεγαλώσει ο Heap πρέπει ρητά να το ζητήσει το πρ. χρήστη από OS
 - *Sbrk* (Set Break point) environment call
 - a7 = 9; a0 = πλήθος αιτουμένων Bytes
 - Επιστρέφει a0 = pointer to 1st allocated Byte
- Στοίβα & Heap μεγ. προς αντίθετες κατ.
 - ⇒ ολική χρήση χώρου
- Ασκ.7: Συνδεδεμ. Λίστα & Διαδικασίες



```
long long int fact( long long int n )
{ if ( n<2 ) { return(1); } else { return( n * fact(n-1) ); } }
```

```
fact:  addi    t0, zero, 2    # immediate 2 needed for "if(n<2)"
      bge    a0, t0, elseF  # if n<2 false, i.e. if n≥2 goto ELSE
      addi   a0, zero, 1    # THEN: create return-value 1, place in reg. a0
      jr     ra             # return --this is the end of the "then" clause
elseF: addi   sp, sp, -16   # PUSH1: allocate 16 Bytes on the stack
      sd     ra, 8(sp)     # PUSH2: save ra into first allocated word
      sd     a0, 0(sp)     # PUSH3: save my argument (n) into second word
      addi   a0, a0, -1    # create argument (n-1) into a0 for my child
      jal    ra, fact      # call my child procedure
      add    t0, a0, zero  # copy return value from my child into t0
                          # (because I need to restore my own argument into a0)
      ld     ra, 8(sp)     # POP1: restore ra from stack
      ld     a0, 0(sp)     # POP2: restore a0 from stack
      addi   sp, sp, 16    # POP3: dealloc the 16 B that I had allocated
      mul    a0, a0, t0    # multiply my own arg a0==n times the return
                          # value from my child that I had copied into t0, and
                          # place the result into a0, as my own return value
      jr     ra             # return
```