Le Développement sous UNIX

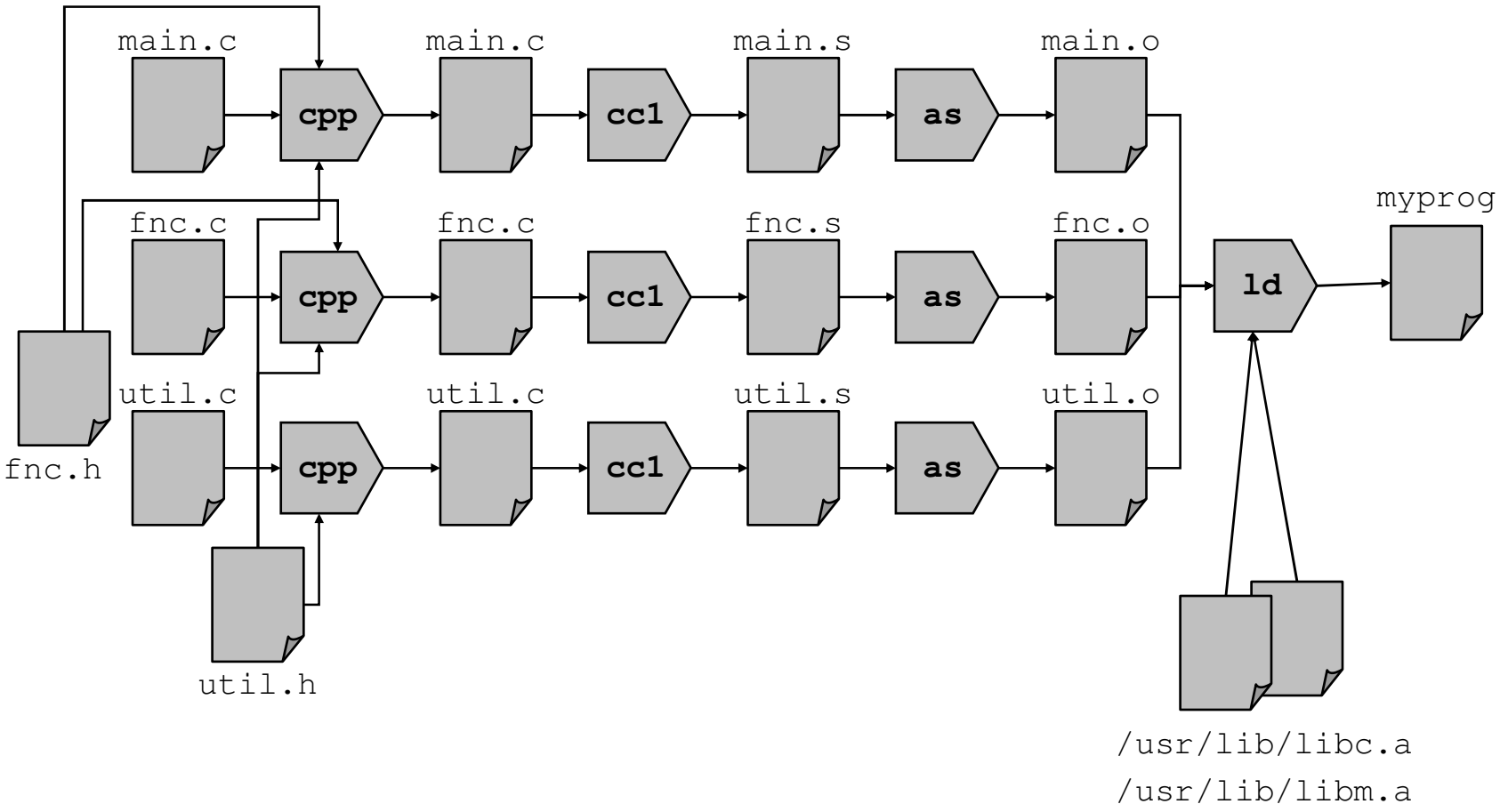
Didier DONSEZ
Université Joseph Fourier (Grenoble 1)
PolyTech'Grenoble – LIG/ADELE
Didier.Donsez@ieee.org
Didier.Donsez@imag.fr

Des outils pour ...

- Organiser
  - Projets et Versions
- Vérifier
  - Tracer et Débogger
- Evaluer
  - Performance
- Documenter et Distribuer

La Chaîne de Compilation i

```
gcc -v -o myprg main.c fnc.c util.c -lm
```



La Chaîne de Compilation ii

- Précompilation (cpp)
  - remplacement des #define
  - inclusions de fichiers de déclaration (.h)
- Compilation
  - production vers un code assembleur (.s)
- Assemblage
  - production d ’un objet (.o)
- Edition de lien (linkage)
  - regroupement des objets et de bibliothèques et résolution des symboles
  - production d ’un exécutable

La Chaîne de Compilation iii

Plus efficace

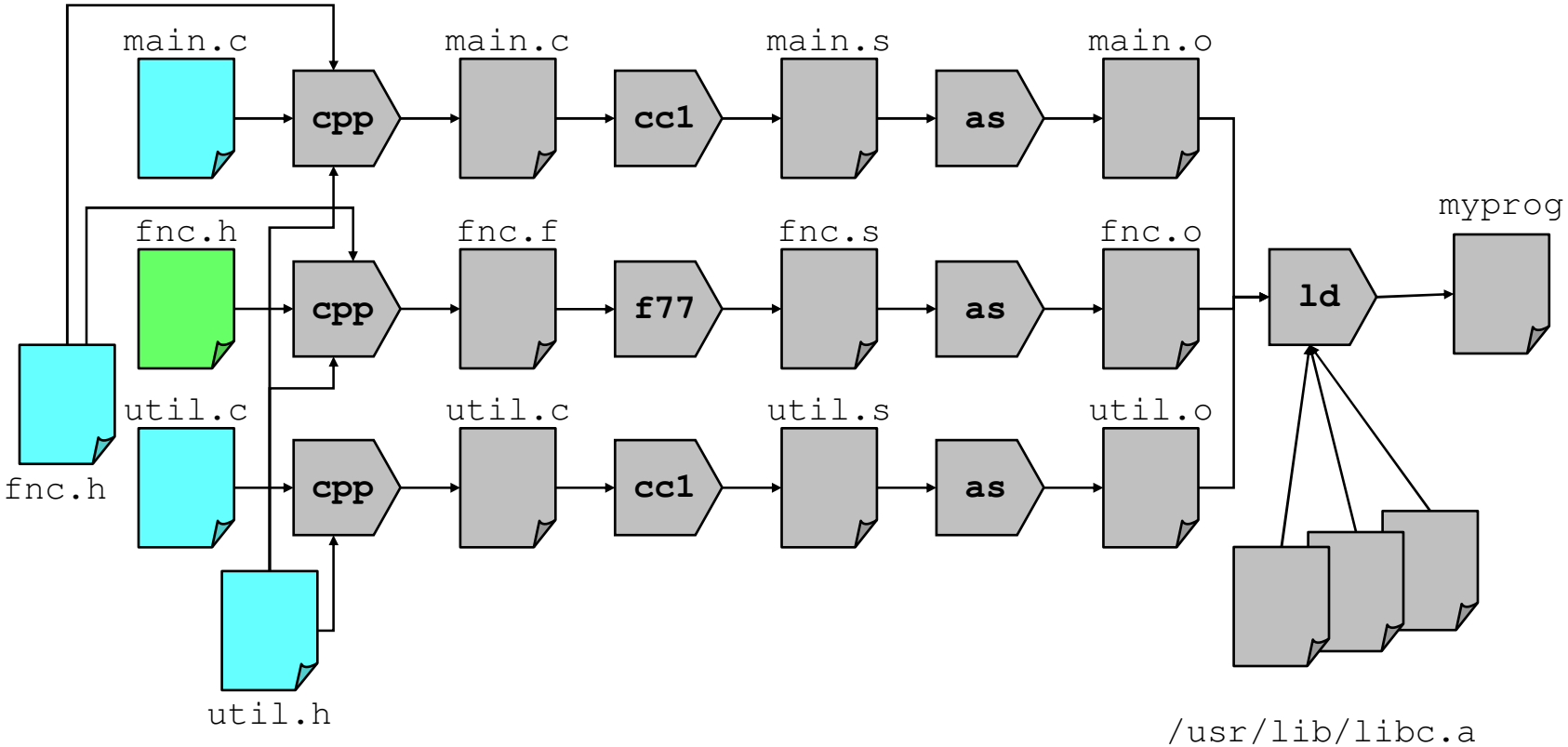
1%	cc -c util.c
2%	cc -c fnc.c
3%	cc -c main.c
4%	ld -o myprg main.o fnc.o util.o -lc -lm

Mais que lancer si

`main.c` est modifié	3 puis 4
`fnc.h` est modifiée	2,3 puis 4
`util.h` est modifiée	1,2,3 puis 4

La Chaîne de Compilation iv
Le développement multilangage

- C, C++, Fortran, Pascal, Perl, Java



La Chaîne de Compilation v

Plusieurs langages

Remarque : production de « stub »

fnc.h des fonctions Fortran de fnc.f

A faire

1%cc -c util.c

2%f77 -c fnc.f

3%cc -c main.c

4%ld -o myprg main.o fnc.o util.o -lc -lm -lf77

La Chaîne de Compilation vi

Rendre un code plus portable

- architecture et systèmes d ’exploitation
  - cross-compilation
- modes
  - normal, déboggage, tracable, monitoré, optimisé
- compilateurs
  - cc/gcc, g++/visualC++, ...

```
cc -c -p util.c
cc -c -O2 util.c
cc -c -g util.c
cc -c -g -target sparc util.c
```

Organiser un projet en sous-répertoires

```
/projet101
   Makefile
   makedepend
   /src
   /include
   /docs
   /man
   /testsrc
   /i86_linux/test
   /i86_linux/obj/norm
   /i86_linux/obj/opt
   /i86_linux/obj/debug
   /i86_linux/bin/norm
   /i86_linux/bin/opt
   /i86_linux/bin/debug
   /i86_sunos5/ ...
   /sparc_sunos5/ ...
   /sparc_sunos4/ ...
```

Organiser (i)

- make
  - gestion des étages de compilation
  - organisation des objets d 'un projet en fonction de
    - des modes de compilation, de la plate-forme, de l 'OS, …
  - principe
    - description des dépendances temporelles et des actions
    - dans un fichier `Makefile`
  - Outils associés
    - `makedepend, imake, autoconf`
- ant, maven
  - utilisable bien qu'orienté Java

Organiser (ii)

- Développement Coopératif (ou Collaboratif)
  - une ou plusieurs équipes de plusieurs développeurs
  - concurrents : i.e. ils travaillent en même temps
- Organisation des développements
  - Découpage de l 'application en plusieurs parties
    - isolation par des interfaces
  - Plusieurs versions de chaque partie
    - versions successives (alpha, beta, prod, …)
    - versions alternatives (langages, algorithmes, stratégies, plusieurs équipes …)
- Gestion des versions de sources
  - SCCS, RCS, ClearCase, Perforce, SourceSafe, VSS, …
  - CVS (the Concurrent Versions System)
    - http://www.cyclic.com/cyclic-pages/howget.html

Tracer

- tracage du développeur *(cpp -DDEBUG)*
  - insertion de lignes de tracage à la Précompilation

```
#ifdef DEBUG
        fprintf(stderr, "Trace 1 ... ", ...);
#endif
```

    - *REM : `cpp` n 'est pas reservé qu 'au C (Fortran, Pascal, ...) et il existe Mocha pour Java*
- ctrace (option -f *fnc*)
  - insertion de ligne de traçage dans un source C
- cflow (option -r -ix)
  - graphe des appels de fonction
- truss (option -t *trap* -s *signal*)
  - tracage des appels système (au noyau)
- dtrace (Solaris 10+), systemtap (Linux)
  - ajout d'agents/sondes d'observation personnalisés

DTrace

- Système de tracage du noyau OpenSolaris
  - Également pour MacOS X, FreeBSD, QNX
  - Ajout et retrait dynamique (OS,pus) de probes
  - Faible impact sur les performances → orienté vers les machines de production
  - *ustack helper* d'interpréteurs Sun JVM, Sun RTJVM, Python, Ruby, PHP, JavaScript@Firefox, …
- Langage D
  - Compilé par DTRACE puis communiqué par la commande `dtrace`
- Command `dtrace`
  - dtrace –n "syscall:::entry {@[execname] = count()}"
  - dtrace –s diskio5m.d
  - …
- GUIs
  - dlight, instruments (MacOS X)…
- Références
  - http://www.opensolaris.org/os/community/dtrace/
  - http://en.wikipedia.org/wiki/DTrace

DTrace: exemples de commandes

```
# New processes with arguments,
dtrace -n 'proc:::exec-success { trace(curpsinfo->pr_psargs); }'

# Files opened by process,
dtrace -n 'syscall::open*:entry { printf("%s %s",execname,copyinstr(arg0)); }'

# Syscall count by program,
dtrace -n 'syscall:::entry { @num[execname] = count(); }'

# Syscall count by syscall,
dtrace -n 'syscall:::entry { @num[probefunc] = count(); }'

# Syscall count by process,
dtrace -n 'syscall:::entry { @num[pid,execname] = count(); }'

# Disk size by process,
dtrace -n 'io:::start { printf("%d %s %d",pid,execname,args[0]->b_bcount); }'

# Pages paged in by process,
dtrace -n 'vminfo:::pgpgin { @pg[execname] = sum(arg0); }‘

# Format
dtrace -n ‘#pragma  D flowindent pid712:::entry;'
```

Sources http://en.wikipedia.org/wiki/DTrace

SystemTap (Linux)

- *“SystemTap provides free software (GPL) infrastructure to simplify the gathering of information about the running Linux system. This assists diagnosis of a performance or functional problem. SystemTap eliminates the need for the developer to go through the tedious and disruptive instrument, recompile, install, and reboot sequence that may be otherwise required to collect data.*

- *SystemTap provides a simple command line interface and scripting language for writing instrumentation for a live running kernel. We are publishing samples, as well as enlarging the internal "tapset" script library to aid reuse and abstraction. We also plan to support probing userspace applications. We are investigating interfacing Systemtap with similar tools such as Frysk, Oprofile and LTT.*

- *Current project members include Red Hat, IBM, Intel, and Hitachi.”*

- *http://sourceware.org/systemtap/documentation.html*

Débogger i

- But
  - contrôler l ’exécution d ’un programme en pas à pas
- Bas Niveau
  - pas = instruction langage machine
  - visualisation des l ’état des registres et de la mémoire
- Symbolique
  - pas = ligne d ’instruction des langages utilisés
    - ils peuvent être plusieurs dans un même exécutable (.c, .f, .p)
  - visualisation des l ’état des variables

Débogger ii

- Post Morten
  - à la « mort » de processus
    - core : fichier image de la mémoire du processus à sa « mort »
- Exécution Pas à Pas
  - lancement du processus
    - surveillance à des points d ’arrêt
    - déroulage du programme en pas à pas
  - cross-déboggage
    - arrêt d ’un processus en cours d ’exécution
    - reprise en pas à pas
    - Remarque : utile pour le déboggage multi-processus

Débogger iii

- Déboggeurs
  - Bas niveau
    - adb
  - Symboliques
    - compilation avec l 'option -g (ajout d 'information de déboggage)
    - déboggage avec sdb, dbx, gdb (pour C, Fortran), jsdebug (JavaScript), jdb (Java)
    - Script de deboggage avec Mdb (http://docs.sun.com/app/docs/doc/816-5041)
  - Noyau (developpement de modules et de pilotes)
    - Kgdb (par une liaison série, JTAG, …)
    - Kmdb (mdb pour le noyau Solaris)
    - Linux in a process
- Principales Fonctionnalités
    - break, watch, set, print, dump, catch, ignore, ...
    - run, step, stop, when, cont, ...
  - Multi-Processus / Multi-Thread
    - déboggage d 'un processus (ou thread) particulier du programme
    - sur une machine distante éventuellement
- GUI
  - « front-end » graphique à un deboggeur en ligne de commande
    - dbxtool, xgdb, ddd, ceux des AGL C++ et Java ...

Debogger iv

- Bug Tracking
  - Motivations
    - faire remonter les bogues rencontrés par les utilisateurs finaux sur le logiciel déployé (donc en opération)
    - Afin que les développeurs puissent corriger le bogue
- Outils: Mantis

Evaluer i

- prof
  - compilation avec l 'option -p
  - exécution et production du fichier mon.out
  - `prof myprg` fournit statistiques et comptages
    - *durée, nombre d 'appels, ...*

Evaluer ii

- time
  - afficher la durée d ’exécution
    - temps réel
    - temps CPU en mode utilisateur
    - temps CPU en mode noyau
  - exemple

```
> time gcc argc.c
0.4u 0.7s 0:01 59% 0+336k 7+33io 3pf+0w
> /usr/bin/time gcc argc.c
        2.0 real         0.4 user         0.7 sys
```

Evaluer iii

- Banc d 'essai (Benchmark)
  - évaluer la performance d 'un système (logiciel+matériel) dans des conditions proches de celles du système en production
  - afin de dimensionner le système (matériel) ...
  - Charge de travail
    - travail (job, requête, transaction, …)
    - composition de travaux types (rencontrés en production)
  - Mesures
    - temps de réponse d 'un travail
    - nombre de travaux par unité de temps (seconde, minute, jour ...)
    - prix du système (OTC : Ownership Total Cost) incluant la maintenance
- Des benchmarks célèbres
  - SPECMark, TPC/A, TPC/D, TPC/W, ...

Optimiser (i)

- optimise les performances d ’un programme
  - améliore le temps CPU en mode utilisateur
- Niveaux d ’optimisation
  - optimisation rendant le programme de plus en plus performant
- Techniques d ’optimisation
  - en fonction du processeur cible (RISC, SuperScalaire, …)
  - affectation de variables, branchement, test, boucle, déroulage de boucle, ...
- Options de compilation
  - -O
  - -O2
  - …

Optimiser (ii) - Remarques

- le mot-clé `volatile` en C supprime l 'optimisation de l 'affectation des variables
  - utiliser pour les périphériques mappées en mémoire

```
volatile int* serialdevice_cmd  = (int*) 0x0210;
volatile int* serialdevice_data = (int*) 0x0214;
*serialdevice_cmd = 0x0C11; *serialdevice_data = 0x15AB;
*serialdevice_cmd = 0x0CA1; *serialdevice_data = 0x76BC;
*serialdevice_cmd = 0x0CA1; *serialdevice_data = 0x43EF;
```

- l 'optimisation allonge le temps de compilation et consomme de la mémoire
  - il faut prévoir d 'optimiser une fois le programme déverminé
- l 'option de déverminage symbolique
  déactive parfois les options d 'optimisation
  - l 'objet généré n 'est plus le même
- Attention : l 'optimiseur est parfois boggé *(vérification avec adb)*

Documenter

- Génération automatique de la documentation
  - à partir des commentaires du source (#, //, /**, …)
- Contenu de la documentation
  - Synopsis et définition des fonctions, SEE ALSO, ...
  - hiérarchie de classes, dépendance de fonctions, ...
- Format
  - man (troff), texinfo
  - HTML, XML (navigation hypertextuel), Win .hlp
- Exemple
  - javadoc, doclets pour Java
  - nombreux AGL C-C++, caml *(voir VP)*, ADA *(voir DG)*, ...

Bibliothèques, Archives et Objets Partagées

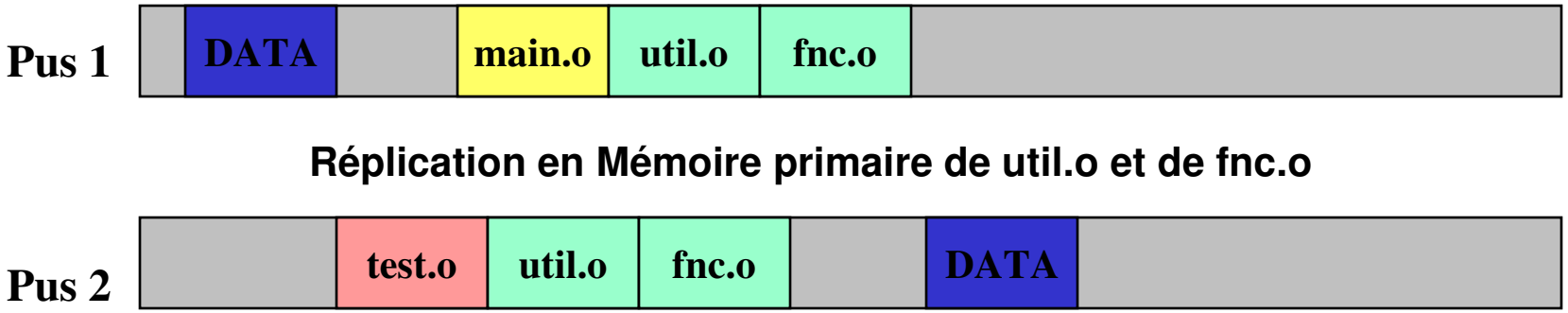
- But

  regrouper des objets utilisables en une seule bibliothèque

- Archive et édition de liens statique
  - plusieurs `.o` dans un seul fichier `.a` (plus pratique)
- Objets Partagés et édition de liens dynamique
  - partage d 'objets `.so` (Shared Object) en mémoire primaire entre plusieurs processus
  - REM : .DLL sur Win32
- Outils d 'inspection des objets
  - nm, size, ldd (liste les dl nécessaires)

Archive et édition de liens statique

```
ar c $HOME/lib/libutil.a util.o fnc.o
ld -o main main.o -lc -L$HOME/lib -lutil
ld -o test test.o -lc -L$HOME/lib -lutil
./test & ; ./main &
```



Objets Partagés (**.so**) et édition de liens dynamique

- partage d 'un segment de code entre plusieurs processus
  - économie de mémoire primaire

ld -G -o $HOME/lib/libutil.so fnc.o util.o

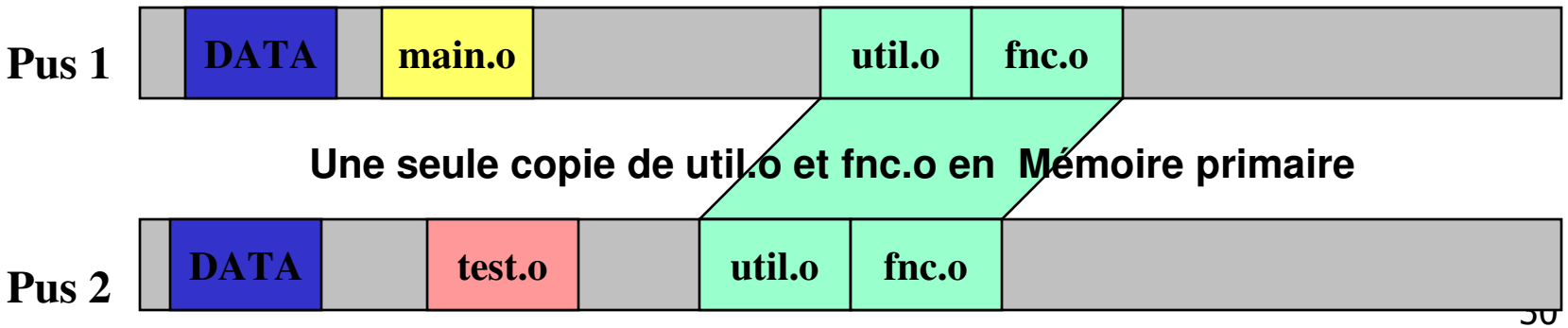
cc -o main main.c -L $HOME/lib -lutil

cc -o test test.c -L $HOME/lib -lutil

setenv LD_LIBRARY_PATH $LD_LIBRARY_PATH:$HOME/lib

./test & ; ./main &

cc **-dn** -o teststatique test.c -L $HOME/lib -lutil



Distribuer = Packager + Installer

- Livraison sous la forme d ’un seul fichier
  - .tar, .tgz, .jar, .shar (script shell), .exe, .zip …
- Installation et Configuration
  - Détection de prérequis
    - logiciels comme make, perl, jre, libraries, matériels, …
  - Choix du répertoire d ’installation
  - Paramêtrage (nom d ’usager, de machine, …)
- Mise à Jour d ’une installation
  - totale/incrémentale
  - procédure de recupération de la version antérieure
    - en cas d ’abandon (panne, …) au cours de la mise à jour

Bibliographie

- Rifflet, « La Programmation sous UNIX », Ed Interedition
- Documentations des outils
- Collections des éditions O 'Reilly
  - make, ...
- Richard McDougall, Jim Mauro, Brendan Gregg, "Solaris Performance And Tools: DTrace And Mdb Techniques for Solaris 10 And OpenSolaris", Editeur : Sun Microsystems Press; Édition : 1 (17 août 2006), ISBN-10: 0131568191, 496 pages.