GenDbg : un débogueur générique

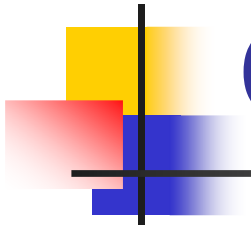
Didier Eymery
Jean-Marie Borello
Jean-Marie Fraygefond
Odile Eymery
Philippe Bion

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE
MINISTÈRE DE LA DÉFENSE

DGA

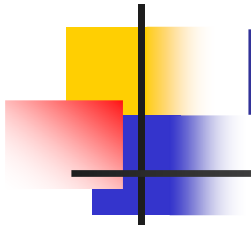
Qui sommes nous ?

- **Centre d'électronique de l'Armement (CELAR)**
  - Maîtrise et protection de l'information
- Division Sécurité des Systèmes d'Information
- Département **Analyse Menace Informatique**



Que faisons nous ?

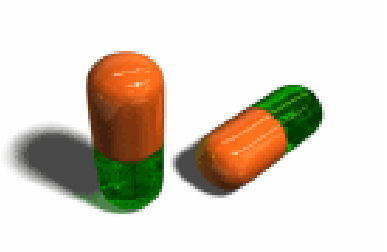
- Analyses de sécurité informatique sur des systèmes de défense
  - Systèmes d'armes
  - Systèmes d'informations
  - Logiciels industriels & commerciaux (COTS)
  - Approche « boîte noire »
    - Pas de codes sources disponibles
    - Pas de spécifications détaillées
    - **Brochures commerciales** ;-)
  - Caractérisation des vulnérabilités et de leurs conditions d'exploitation
  - Connaissance de la menace (« les méchants »)



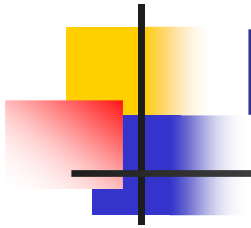
Débogueur ?

- Logiciel permettant l'analyse dynamique de code machine
  - État des registres & de la mémoire
  - Exécution pas à pas des instructions
- Utilisé principalement
  - Lors de la phase de mise au point des programmes …et des exploits
  - Lors de l'analyse d'un système fermé

Générique ?

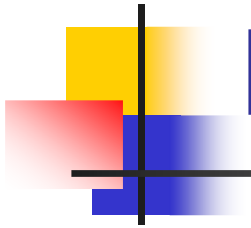


- C’est-à-dire non lié à
  - Une Plateforme
    - PC, Mac, SmartPhone, PocketPC…
  - Une Architecture
    - IA-32, IA-32e, IA-64, ARM, PowerPC…
  - Un Système d’exploitation
    - Windows NT, Linux, IOS, WinCE, Symbian…



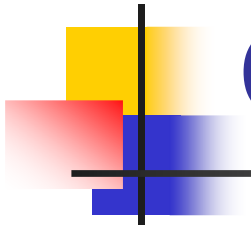
Motivations

- De plus en plus d'informatique embarquée & communicante
- Architectures hétérogènes
- Technologies civiles intégrées dans des systèmes militaires (hard et soft)
- Continuer à évaluer ces systèmes avec les méthodes « AMI »
  - Outillage



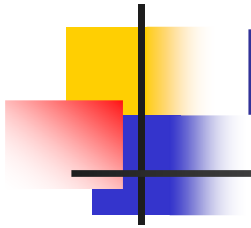
Motivations

- Chaque environnement de développement propose son propre débogueur
  - IHMs, APIs, Syntaxes, mises en œuvre différentes à chaque fois
- Solutions non pérennes
  - Ex: Softice
- Solutions essentiellement PC/Windows/Intel
  - OllyDbg (Win32), WinDbg…
- Solutions « génériques » mais pas trop…
  - Gdb, Rasta Ring0 (Intel)



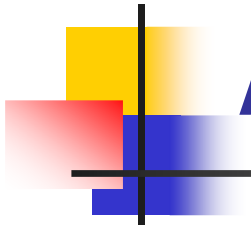
GenDbg : nos besoins

- Besoins liés à l'outil
  - Maîtrise
    - APIs, code (ce sont NOS bugs!)
    - Maintien du niveau d'expertise
- Modularité
  - Extensibilité, développement collaboratif
  - Généricité
    - Plateforme, architecture, OS



Fonctionnalités et choix

- IHM mode texte « à la softice »
  - Efficacité
- Moteur de script « maison »
  - Réutilisation, cohérence avec d'autres outils
- Architecture client/serveur ouverte
  - Modularité
  - Travail sur machine distante
  - Notion de « stubs » empruntée à Gdb
  - Protocole de communication simple

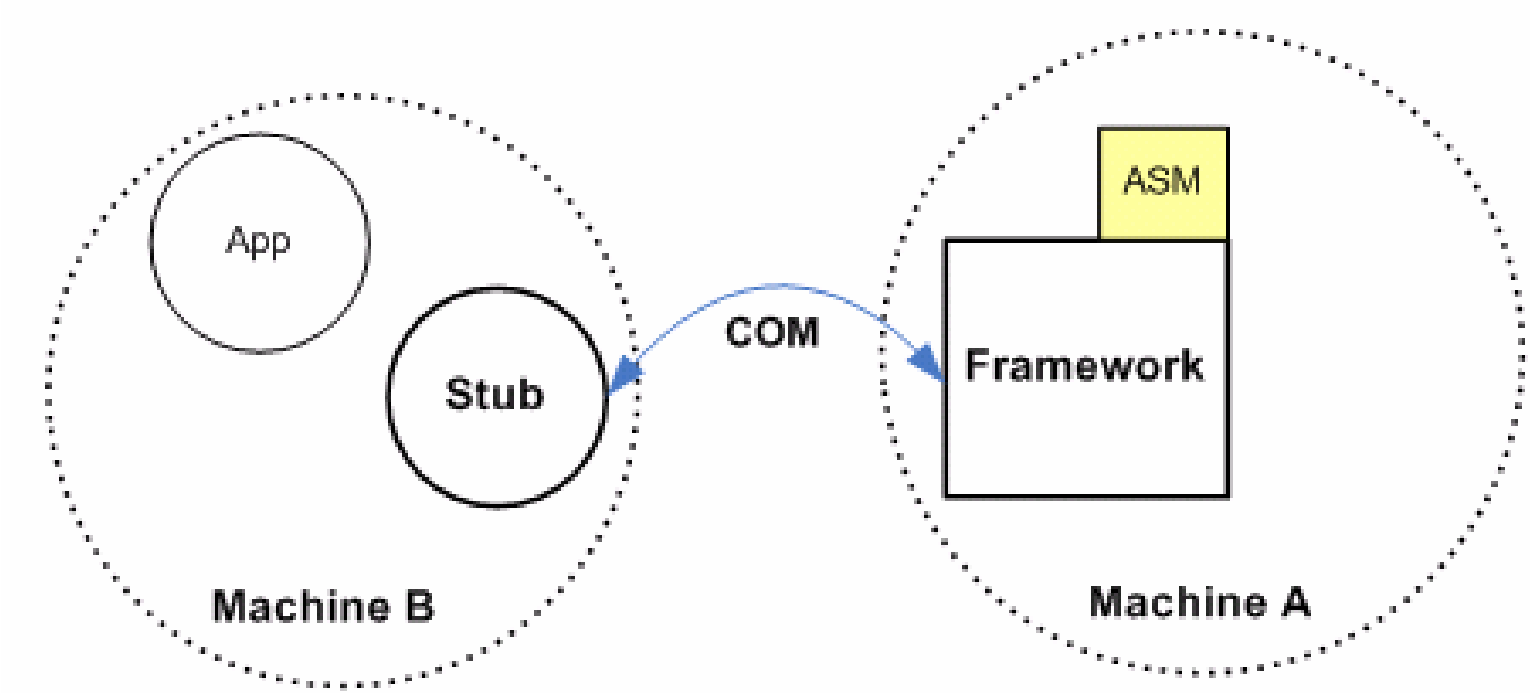


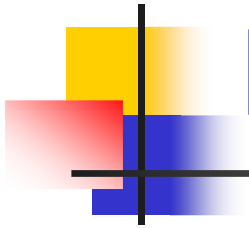
Abstractions logicielles

- Programmation C/ASM mais approche objet pour la conception
  - Les modules manipulent des types abstraits (*structures de données*) via des méthodes (*fonctions*) fournies par le framework et/ou d'autres modules
    - Ex de données structurées:
      - Adresses mémoire, Contexte d'exécution, Datas
    - Ex de méthode:
      - AddressToTxt(…)
      - Adresse virtuelle → « 0008:40200000 »

Architecture Minimale

- Trois composants logiciels nécessaires et suffisants : Framework + Stub + module ASM



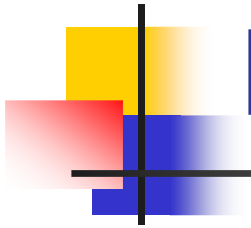


Le Framework

- Implémenté sous la forme d'un binaire Win32.
- Gère l'IHM
  - « softice like »
- Les communications inter-modules
  - Callbacks
  - Helpers functions
- La communication avec les Stubs
  - Protocole GenDbg
    - Liaison série, PIPE
- Moteur de script

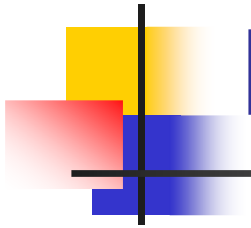
Stubs

- « Têtes de pont »
- Le seul code devant s'éxecuter sur la machine cible
- Fortement liés à l'OS et/ou à l'Architecture
  - Gestion d'interruptions/exceptions
- Les stubs implémentent des primitives:
  - Lecture/écriture de registres
  - Lecture/écriture de la mémoire
  - Fonctions privées
- Le stub identifie la cible
  - Plateforme, Architecture, OS
- Communication série



Modules ASM

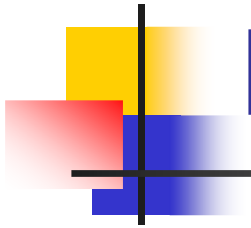
- Désassembleurs/Assembleurs
- Liés à une architecture
  - Ex: IA-32
- En charge de la représentation
  - Des Registres
  - Du Code (Mnémoniques)
  - Données
  - Adresses mémoire



Modules complémentaires

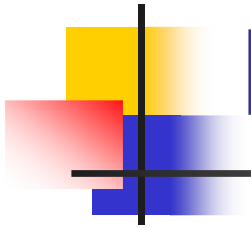
- A cette architecture « minimale » peuvent s'ajouter des modules optionnels :
  - Modules de Commandes
  - Modules OS Helpers
  - Modules de Symboles
  - Modules de BreakPoints

- Optionnels…mais néanmoins bien utiles!
  - PROC, MOD, DRIVER, DEVICE…



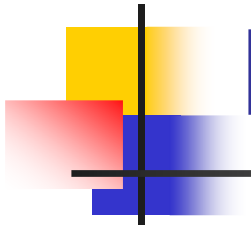
Modules CMD

- Modules de commandes additionnelles
- Enrichissent le jeu de commandes de base inclu dans le framework
- Ex:
  - les commandes IDT, GDT, TSS… sont implémentées dans un module de commande lié à l'architecture IA-32



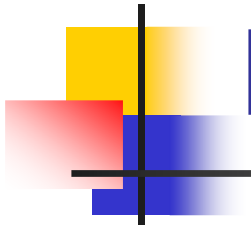
Modules OS Helpers

- Offrent des services et des fonctionnalités propres à l'OS cible
  - Signalisation d'évènements
    - Chargement/déchargement de module
    - Création/destruction de processus/thread
  - GetModuleBase(…)
  - GetAddressInfo(…)
    - Teste l'appartenance d'une adresse à un module



Modules SYM

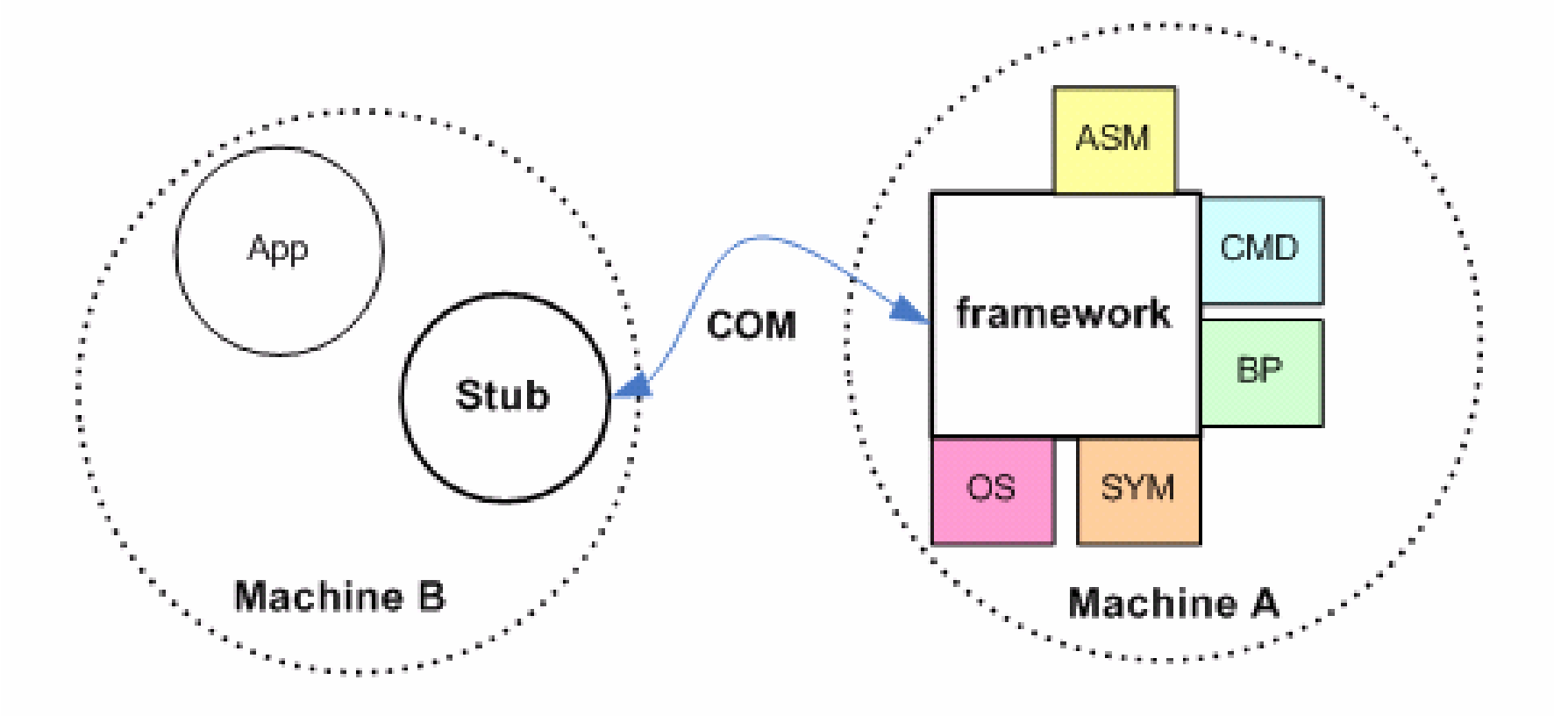
- Ces modules permettent l'association d'adresses et de symboles.
- Ok ?



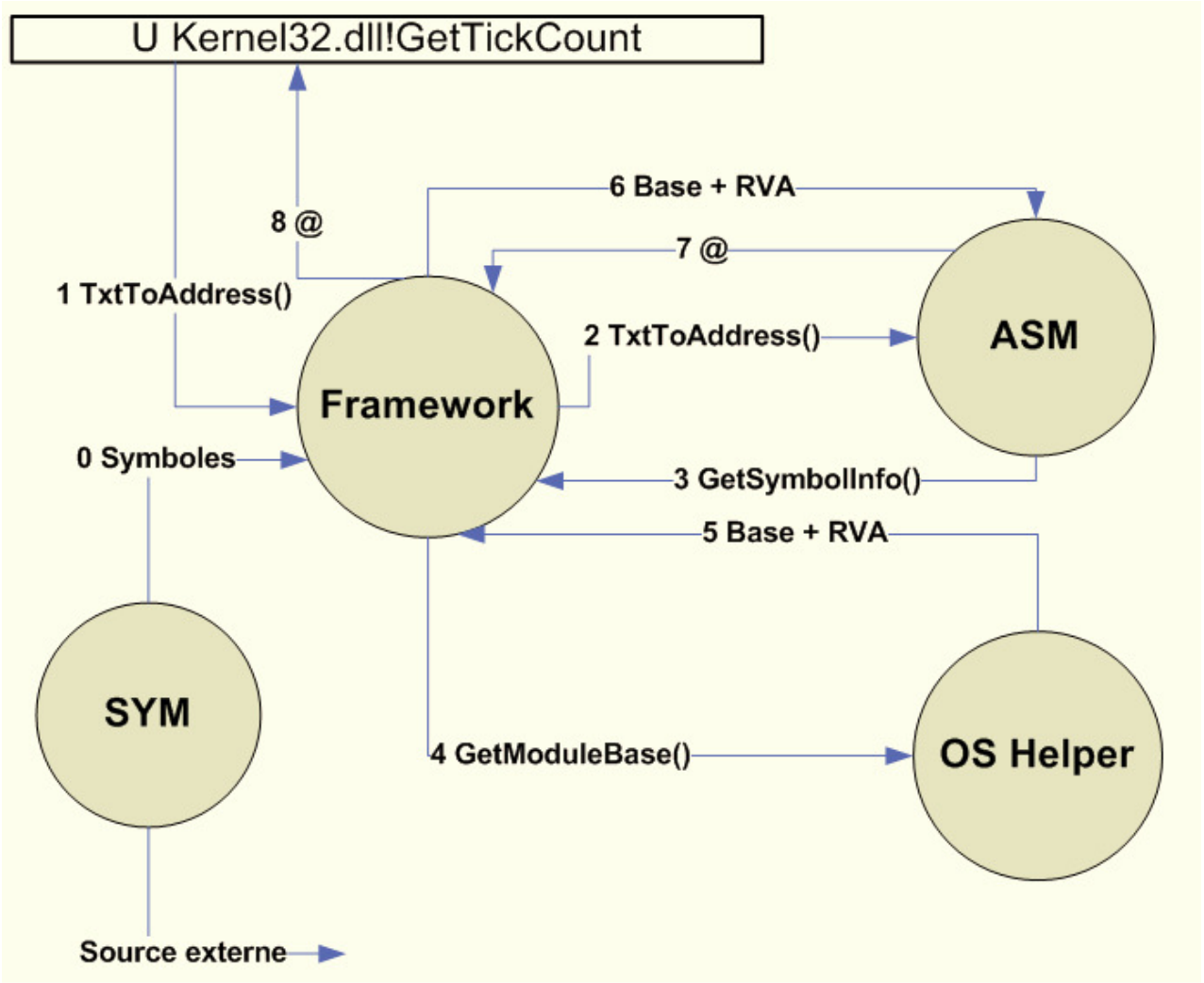
Modules BP

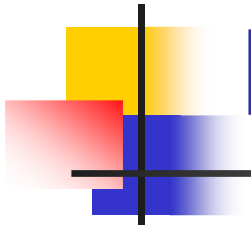
- Breakpoint Extenders
  - Ces modules ajoutent des fonctionnalités pour positionner des points d'arrêts dans le flot d'exécution.
  - S'appuient sur les spécificités de l'architecture cible
    - Ex: prise en charge des BPM sur Intel

Architecture Complète



Tuyauterie

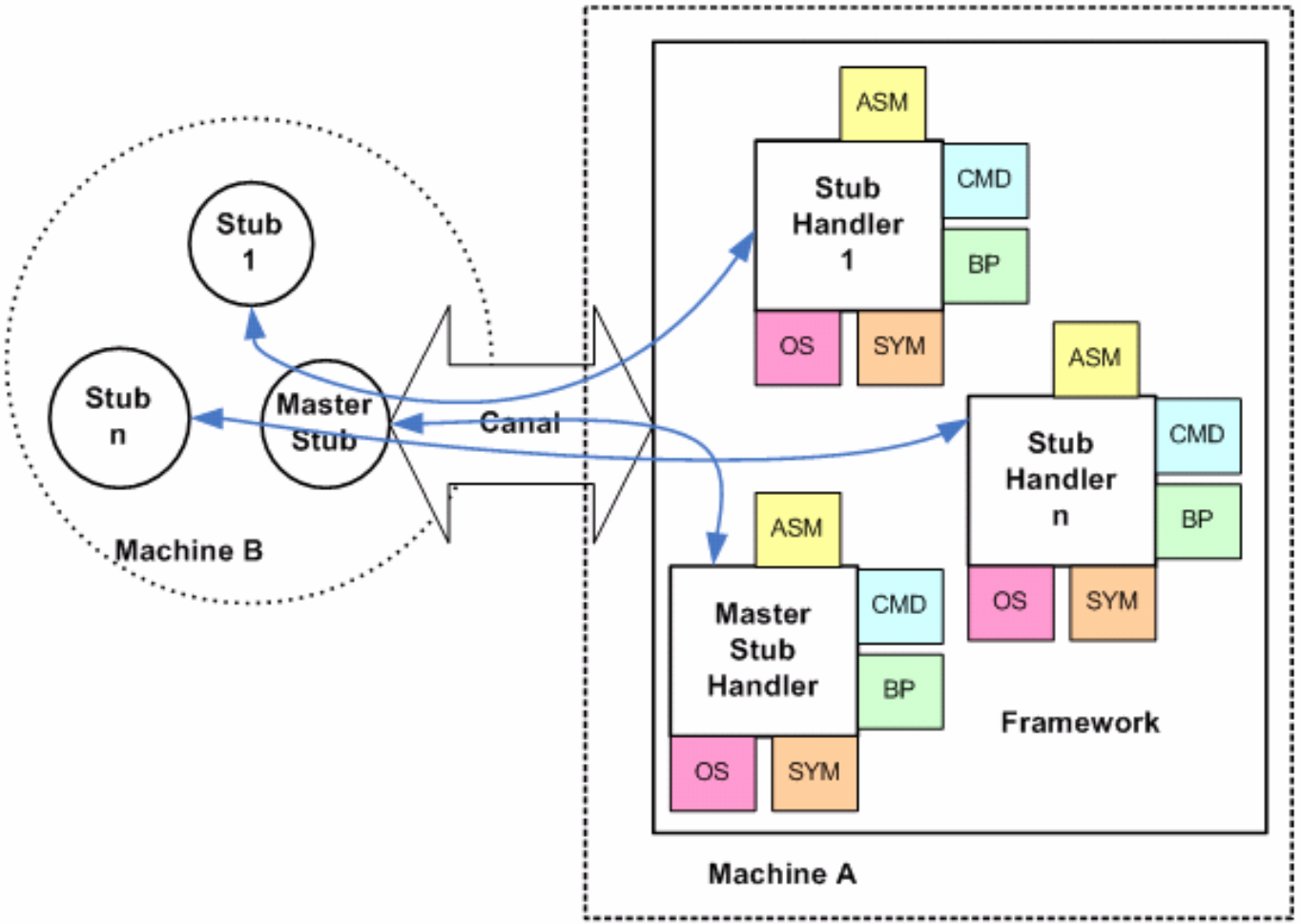

U Kernel32.dll!GetTickCount
6 Base + RVA
8 @
7 @
1 TxtToAddress()
2 TxtToAddress()
ASM
Framework
0 Symboles
3 GetSymbolInfo()
5 Base + RVA
SYM
4 GetModuleBase()
OS Helper
Source externe



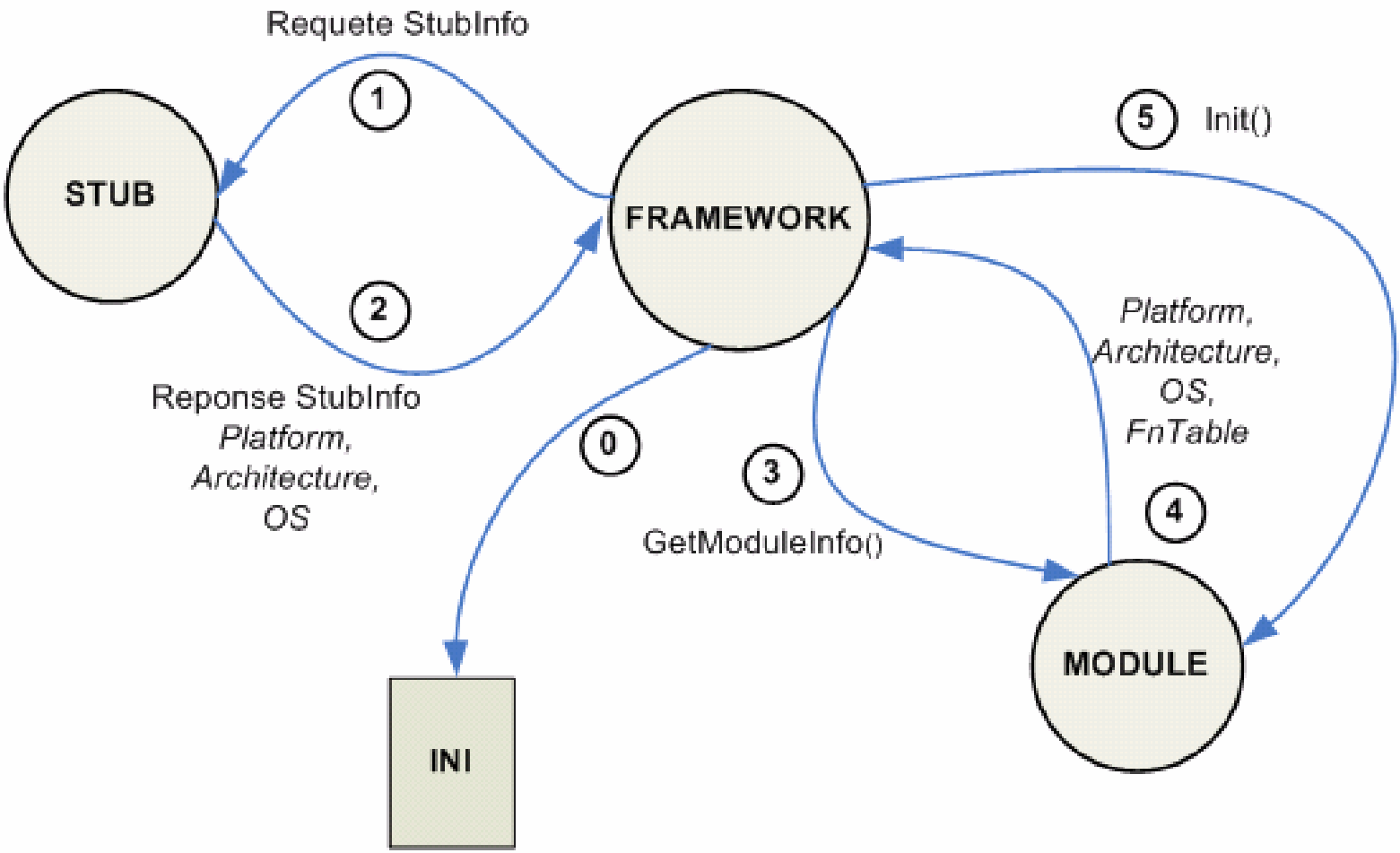
MultiStub

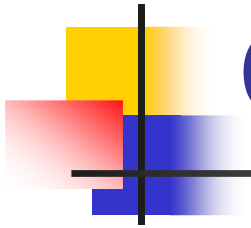
- GenDbg peut utiliser plusieurs stubs pour déboguer une même application.
- Utile pour les machines virtuelles (*java*) ou les langages interprétés
- Permet d'avoir plusieurs niveaux de « visibilité »
  - Ex:
    - un stub trace le code interprété tandis qu'un autre peut tracer le code natif de l'OS ou de la machine virtuelle

Architecture complète multistub



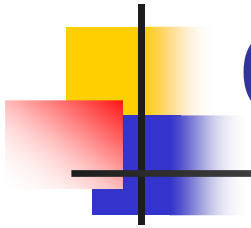
GenDbg : initialisation





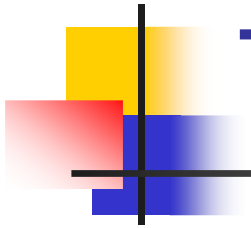
GenDbg aujourd'hui

- Un framework Win32
  - Gendbg.c = 480ko = 12000 lignes de C
- 12 Stubs
  - Win32 (Debug API)/Intel , Linux/Intel Ring0, WinNT/Intel Ring0, Java, .NET, VB (sous stub), WinCE/ARM, Wrapper Gdb, BIOS (Intel mode réel), Linux/Intel Ring 3 (Ptrace API)
- 50 Modules
  - 6 Modules ASM
    - PPC, ARM, Java, .NET, PCODE VB, x86 32 bits
  - 5 Modules OS Helpers
    - WinNT, Win32, Java, BIOS, WinCE
  - 11 Modules de Commandes
    - Generic IA-32, WinNT, Generic ARM…
  - 3 Modules de Symboles
    - PE, PDB, BIOS
  - 2 Modules Breakpoint Extenders
    - Gestion BPM Intel …
- 5 développeurs ☺



C’est disponible ?

- Les spécifications du framework sont documentées
  - SDK
    - Gendbg.h & co ☺
    - Définitions des interfaces de programmation
    - Définitions des types de données
- L’implémentation du framework peut être distribuée sous forme binaire
  - Stub Win32 + Jeu de modules de base
    - Tests & évaluations



TO DO List…

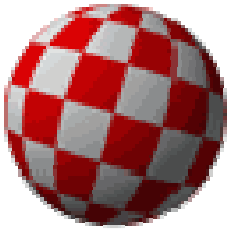
- OS Helpers
  - Travail important pour les OS « fermés »
    - Connaissance approfondies des mécanismes internes de l'OS
  - NT
    - support multi cœurs
    - Support architecture IA-64 et IA-32e
- Stubs
- Modules de commandes
- Scripts

Questions ?

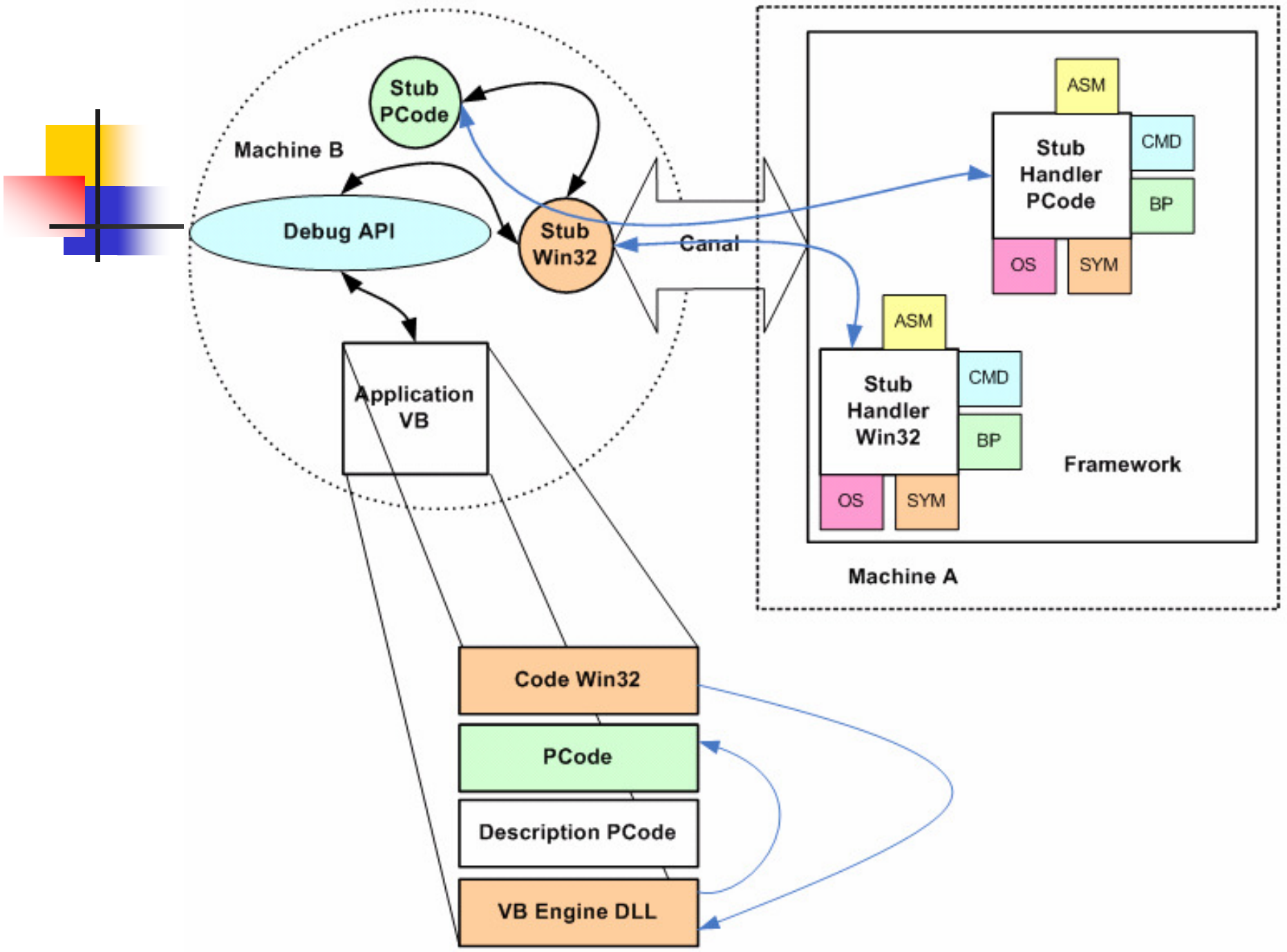
Contact :

- jean-marie.fraygefond@dga.defense.gouv.fr

Démo multistub



- Application Visual Basic PCode
- Stub Win32 + sous-stub VB
  - L'interpréteur VB (MSVBM60.DLL) est vu comme un microprocesseur virtuel par le sous-stub VB → simulation d'exceptions
- Framework
- Attention ça pique un peu les yeux…


Stub
PCode
Machine B
Debug API
Stub
Win32
Canal
Application
VB
ASM
Stub
Handler
PCode
CMD
BP
OS
SYM
ASM
Stub
Handler
Win32
CMD
BP
OS
SYM
Framework
Machine A
Code Win32
PCode
Description PCode
VB Engine DLL